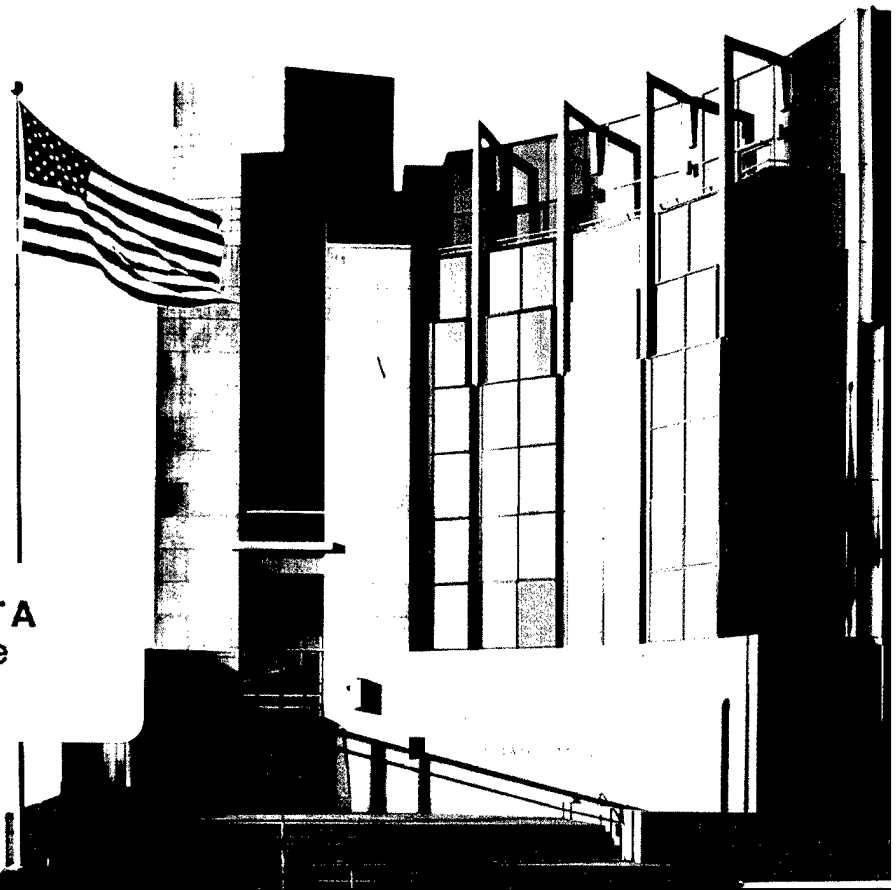
Carnegie Mellon
Software Engineering Institute

Performance Property Theories for Predictable Assembly from Certifiable Components (PACC)

Scott Hissam
Software Engineering Institute
Mark Klein
Software Engineering Institute
John Lehoczky
Department of Statistics,
Carnegie Mellon University
Paulo Merson
Software Engineering Institute
Gabriel Moreno
Software Engineering Institute
Kurt Wallnau
Software Engineering Institute

September 2004



TECHNICAL REPORT
CMU/SEI-2004-TR-017
ESC-TR-2004-017

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Performance Property Theories for Predictable Assembly from Certifiable Components (PACC)

CMU/SEI-2004-TR-017
ESC-TR-2004-017

Scott Hissam
Software Engineering Institute
Mark Klein
Software Engineering Institute
John Lehoczky
Department of Statistics, Carnegie Mellon University
Paulo Merson
Software Engineering Institute
Gabriel Moreno
Software Engineering Institute
Kurt Wallnau
Software Engineering Institute

September 2004

Predictable Assembly from Certifiable Components Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Abstract vii

1 Introduction 1

2 Performance Approaches 3
2.1 Basic Notation 3
2.2 Generalized Rate Monotonic Analysis (GRMA) 4
2.3 Classical Queueing Theory 6

3 Sporadic Servers 9

4 Reasoning About the Average Latency of Aperiodic Tasks Managed by Application-Level Sporadic Servers 15
4.1 Observations About Average Latency When Using a Sporadic Server 16
4.2 Special Case of No Periodics 17
4.3 Special Case of No Background 18
4.4 Special Case of Continuous Background 19
4.4.1 Computing Average Queueing Time (E[Q]) 21
4.4.2 Computing Average Service Time (E[S_s]) 22
4.4.3 Areas of Ongoing Work 36
4.4.4 Empirical Evidence 37
4.5 Single-Subtask Assemblies 40
4.6 Multi-Subtask Assemblies 42
4.7 Observations on the No-Background Case 43

5 Application of the Theory 45
5.1 Reasoning Heuristics 45
5.2 A Robotics-Based Model Problem 47
5.2.1 Tasks in the Model Problem 47
5.2.2 Analysis Setup 48
5.3 Preserving Periodic Deadlines 50
5.4 Predicting Average-Case Latency 51

6 Conclusions 55

6.1 Future Work 55

Bibliography **57**

List of Figures

Figure 1: Average Latency for a Simple Queue 7

Figure 2: Example of a Sporadic-Server-Controlled Task 10

Figure 3: Pseudocode for SStask 11

Figure 4: Pseudocode for SSmanager.request() and SSmanager.arm() 12

Figure 5: Pseudocode for SSmanager.replenishment_timer() 12

Figure 6: UML 2.0 Sequence Diagram of Application-Level SSSA: Request and Arm 13

Figure 7: UML 2.0 Sequence Diagram of Application-Level SSSA: Replenishment 14

Figure 8: $E[W] = f(T_p, U_p)$ for T_a=200, S_a=S_{ss}=10, and T_{ss}=100 16

Figure 9: Sample Timeline 20

Figure 10: Predicting E[Q] 22

Figure 11: Differing Service Times for the Aperiodic Arrivals 22

Figure 12: Histogram of Tr 24

Figure 13: Time to Replenishment and Busy Periods 25

Figure 14: CDF for X_i 28

Figure 15: Tr Blackout Dependency on Previous Blackout 37

Figure 16: $f_{Tr}(t)$ Predicted Versus that Observed Through Simulation 38

Figure 17: E[W] Predicted Versus that Observed Through Simulation 39

Figure 18: E[Q] Predicted Versus that Observed Through Simulation 39

Figure 19: $E[S_s]$ Predicted Versus that Observed Through Simulation 40

Figure 20: Multi-Periodic Example—Utilization Evenly Divided 41

Figure 21: Multi-Periodic Example—Utilization Unevenly Divided 42

Figure 22: Multi-Periodic Example with Multiple Subtasks 42

Figure 23: Heuristics Applied to the Curves 46

Figure 24: Tasks in the Robotics Model Problem 48

Figure 25: Analytic Representation of the Robotics Model Problem 50

Figure 26: Latency Observed for the Model Problem for Various T_p Values 53

List of Tables

Table 1: Basic Notation 4

Table 2: Performance Description of Model Problem Tasks 48

Table 3: Comparison of Prediction Heuristics and Simulation Curves 53

Table 4: Predicted and Actual Average-Case Latency for Task M 54

Abstract

This report develops a queueing-theoretic solution to predict, for a real-time system, the average-case latency of aperiodic tasks managed by a sporadic server. The report applies this theory to a model problem drawn in the domain of industrial robot control. In this model problem, a controller with hard periodic deadlines is "open" to third-party plug-in extensions. The sporadic server is used to limit the invasiveness of aperiodic tasks on the controller's hard deadlines. The theory developed in this report is used to predict the average-case latency of a plug-in managed by a sporadic server.

1 Introduction

The goal of the Predictable Assembly from Certifiable Components (PACC) Initiative at the Carnegie Mellon® Software Engineering Institute (SEI) is to enable the construction of software systems from components in a manner that allows for automatic prediction of system behavior [Wallnau 03]. This goal is realized by developing and enhancing component technologies, using and extending property theories, and developing prototype tools and methods. This report focuses on extending property theories for performance—one of the quality attributes for which the PACC Initiative is developing a prediction capability.

Performance, or specifically timing behavior, is important to all systems. For some systems, ensuring the satisfaction of hard deadlines is of primary importance. For other systems, missing deadlines occasionally is acceptable, provided the miss rate is guaranteed to not exceed a specified threshold. Finally, for some systems, satisfactory performance is defined as meeting average latency requirements. Our ultimate goal is to have prediction capability for all these types of systems.

The initial work in creating a performance property theory (called λ_{ABA}) for a prediction-enabled component technology (PECT) was documented by Hissam and colleagues [Hissam 02]. λ is short for latency, and ABA is short for Average-case, with Blocking and allowing for Asynchrony.

λ_{ABA} is built on a body of work known as Generalized Rate Monotonic Analysis (GRMA) [Klein 93], which offers the ability to predict worst-case latency to ensure that hard deadlines are met. λ_{ABA} extends GRMA to predict average-case latency. λ_{ABA} is constrained to a set of component assemblies whose interpretation[1] reduces to sequences of tasks (unit of concurrency) initiated periodically using both synchronous (e.g., call/return) and asynchronous (e.g., message-passing) communication.

The focus of this report is to generalize the λ_{ABA} theory to include tasks that are initiated stochastically (or aperiodically—the terms are used synonymously) in addition to periodically. This work entails developing a new analytic theory for predicting the average latency of ape-

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1 Interpretation assigns to an assembly specification a meaning—or semantics—in some property theory. Interpretation is described in detail by Wallnau [Wallnau 03].

riodic events in the context of a collection of real-time periodic events. This generalization was motivated by systems that must handle aperiodic events without sacrificing the hard real-time deadlines of periodic processes or tasks.[2] This new theory also imposes analytic constraints—in particular, systems must manage aperiodic events with sporadic servers.

The sporadic server scheduling algorithm (SSSA) [Sprunt 89] was invented to solve the problem of protecting periodic events with hard deadlines from bursts of high-priority stochastic events while giving high priority to processing stochastic events. The hallmark of a sporadic server is that it provides a periodic "virtual processor" within which aperiodic events can be processed and analyzed. This report provides a queueing-theoretic foundation for analyzing the average-case latency of aperiodic events handled by a sporadic server.

Section 2 provides thumbnail sketches of GRMA and queueing theory, which serve as building blocks for this work. In Section 3, we describe sporadic servers in more detail. In Section 4, we develop a new property theory for predicting the average-case latency of aperiodic events serviced under the control of a sporadic server. In Section 5, we describe a model problem from the domain of robotics and show how to apply the property theory to that problem. We conclude in Section 6 with a brief description of where this work is headed.

[2] The terms *task* and *process* are equivalent for the purpose of this report.

2 Performance Approaches

As described by Wallnau [Wallnau 03], a PECT comprises a construction framework and one or more reasoning frameworks. Basically, a construction framework provides a vehicle for specifying a set of assemblies that are automatically analyzable. For an assembly to be analyzed, it is interpreted and its interpretation is evaluated. *Interpretation* is the process of translating an assembly into a property-theory-specific representation suitable for evaluation via logical rules of inference and/or simulation.

In the following subsections, we describe the notation and assumptions of the property theories we are using and developing.

2.1 Basic Notation

We assume a single processor executing a set of periodic tasks and a single aperiodic task. Each task is stimulated to execute by the arrival of a sequence of events (either generated externally such as by the arrival of a message or internally such as by a clock interrupt). When the events for a specific task arrive at regular intervals, that task is designated as periodic with a period of T_i (or T_p when there is only one periodic task). When the arrivals are not periodic, they are aperiodic (or sometimes we say stochastic or random). In this case, the average interarrival interval is denoted by T_a.

Each task (regardless of whether it's periodic or aperiodic) executes for a constant amount of time when stimulated. This time is denoted by S_i, (or S_p) for the periodic task and S_a for the aperiodic task. We refer to this time as the *execution time* or *service time*.

Over a long period of time, each task uses a portion of the processor. For periodic tasks, this usage is usually referred to as *utilization* and denoted by U_i (or U_p), where $U_i = S_i/T_i$. For aperiodic tasks, this usage is referred to as the task's traffic intensity and denoted by ρ where $\rho=S_a/T_a$. (While traffic intensity usually refers to all tasks, $\rho=S_a/T_a$ is referring to the traffic intensity of a single task.)

Later, we describe a special mechanism for scheduling the execution of aperiodic tasks—the sporadic server. The sporadic server is characterized by two parameters: an execution budget and a replenishment period. The execution budget is denoted by S_{ss}, and the replenishment period is denoted by T_{ss}.

Each task also has a latency associated with it. The latency (or waiting time) is how long it takes to complete the servicing of an event in the face of preemption from higher priority events and the queueing time due to prior events. For periodic tasks, we might be concerned with either worst-case or average-case latency. For aperiodic tasks, we are concerned with average latency. In this report, we denote latency with the random variable W and average latency as E[W].

Table 1: Basic Notation

	Aperiodic	Periodic	Sporadic Server
Interarrival time	T_a	T_p or T_i	T_{ss}
Execution time	S_a	S_p or S_i	S_{ss}
Traffic intensity/ Utilization	ρ	U_p or U_i	

The next two subsections offer thumbnail sketches of GRMA and queueing theory.

2.2 Generalized Rate Monotonic Analysis (GRMA)

GRMA is a theory for predicting the worst-case latency of a collection of hard real-time tasks. Rate monotonic analysis (RMA) grew out of the fixed-priority scheduling theory of periodic tasks. The term *rate monotonic* originated as a name for the optimal task priority assignment in which higher priorities are accorded to tasks that execute at higher rates (that is, as a monotonic function of rate). Rate monotonic scheduling is a term used in reference to fixed-priority task scheduling that uses a rate monotonic prioritization. The original theory was subsequently generalized to the point of being practicable for a large range of realistic situations encountered in the design and analysis of real-time systems and is now referred to as GRMA, a codification of which is discussed by Klein and colleagues [Klein 93]. Basic GRMA problems[3] have the following characteristics:

- They involve a collection of periodic tasks executing on a single central processing unit (CPU). In the simplest case, each task has a period, a priority, and an execution time.
- They use priority-based preemptive scheduling.
- Tasks may synchronize to use a shared resource.
- Deadlines are assumed to be at the end of the task's period.
- In some cases, tasks may be broken down into a sequence of subtasks. The entire sequence is initiated periodically; however, each subtask has its own execution time and priority.

[3] These problems are basic in the sense that GRMA can handle a wider class of problem than is characterized here.

Several Key GRMA Results. One of the principal goals of GRMA is to calculate worst-case latency, which can then be compared with a deadline to determine whether it can be met. Computing worst-case latency requires knowing that zero-phasing[4] produces the worst-case latency. The worst-case latency for a task occurs when all the high-priority tasks become ready to execute at the same instant as the given task. This moment is known as the critical instant. The following recursive formula, Equation (1), can be used to compute the worst-case latency of a task i, given that

- Tasks 1 to i-1 are higher priority.
- There is no task synchronization.
- Tasks complete before the end of their periods.
- Task i starts at its critical instant.

$$x_{k+1} = \sum_{j=1}^{i-1} \left\lceil \frac{x_k}{T_j} \right\rceil S_j + S_i \qquad (1)$$

The recursion can be started by setting x_0 to S_i, and it ends when a fixed point is reached—that is, when two successive iterations yield the same result. Variations of Equation (1) can be used to account for blocking, to handle computation past the end of the period, and to handle tasks with multiple subtasks.

Using GRMA for Average-Case Latency. Strictly speaking, GRMA was developed as a worst-case analysis tool; attention was focused on determining conditions leading to worst-case latency. However, λ_{ABA} has used GRMA as an average-case analysis tool. To understand average-case latency, we need some additional terminology:

- *hyperperiod* - The hyperperiod of task i is the least common multiple (LCM) of the periods of all tasks that have a priority greater than or equal to task i's. After a hyperperiod, the pattern of execution repeats.
- *job* - corresponds to each instance of a task's execution during the hyperperiod
- *job latency* - the time it takes from the moment the task is ready to run to the moment it finishes executing. Different jobs under the same task might have different latencies.

Since the pattern of execution is completely defined by a task's hyperperiod, the average latency of task i can be determined by computing the average job latency of all the jobs in task i's hyperperiod. A variation of Equation (1) can be used to calculate the job latency for each job in the hyperperiod; these job latencies can then be used to compute the average latency.

[4] We say two or more events are zero-phased in time when they happen at the same moment (i.e., there is no time delay between them).

2.3 Classical Queueing Theory

Whereas GRMA focuses on the case in which interarrival and service times are deterministic (or at least bounded), queueing theory focuses on the case in which interarrivals and service times are stochastic. Basic queueing[5] problems have the following characteristics:

- Customers (that is, events) arrive to a service facility. They could be messages arriving to a CPU according to a probability distribution. Due to its analytical tractability, exponential interarrival times are often used. We limit ourselves to exponential distribution for now.
- Each customer requires a certain amount of time in the service facility also described by a probability distribution. We limit ourselves to a constant amount of time for now.
- In many cases, queueing models include more than one service facility. We limit ourselves to one service facility (that is, a single CPU).

Key Queueing-Theory Result. The key queueing result that we draw on is the following formula:

$$E[W] = \left(\frac{\rho}{1-\rho}\right)\left(\frac{E[S_a{}^2]}{2E[S_a]}\right) + E[S_a] \tag{2}$$

The first term in Equation (2) above (known as the Pollacek-Khinchin expression [Kleinrock 75]) is the mean queueing time, which we denote by E[Q]:

$$E[Q] = \left(\frac{\rho}{1-\rho}\right)\left(\frac{E[S_a{}^2]}{2E[S_a]}\right) \tag{3}$$

Therefore, Equation (2) basically says that the mean latency is the mean queueing time plus the mean service time: $E[W] = E[Q] + E[S_a]$.

The graph in Figure 1 shows E[W] as a function of ρ (for the case in which $E[T_a] = 200$ and $E[S_a]$ varies from 0 to 190). Notice that for low values of ρ (and $E[S_a]$), $E[S_a]$ is the main contributor to average latency. However, as ρ increases, the dominant term becomes E[Q].

[5] Queueing theory comprises a vast and rich body of knowledge. We describe only the most basic queueing-theory situations.

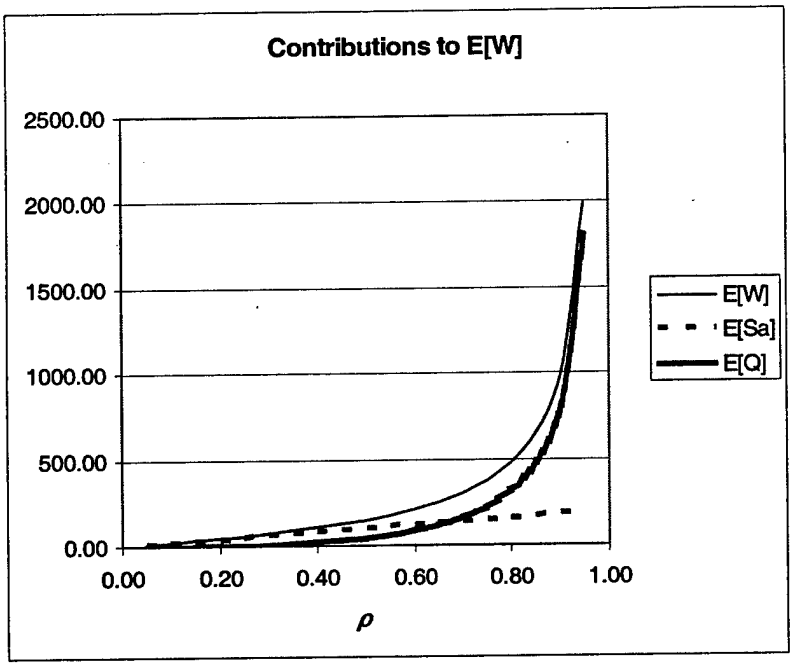


Figure 1: Average Latency for a Simple Queue

A major factor in a queueing system is the queueing time. Queues fill up due to bursts of arrivals caused by variability in arrival and service times.

3 Sporadic Servers

The SSSA [Sprunt 89] was invented to protect periodic events with hard deadlines from bursts of high-priority stochastic events while giving high priority to processing stochastic events. The SSSA both limits and guarantees a certain amount of execution time for aperiodic requests with soft or hard deadlines in a hard real-time system [Gonzalez Harbour 91].

Implementations of the SSSA are based on the general premise that a server (a process within an operating system (OS) or a thread of control within a process) that handles high-priority stochastic events will execute at one of two priorities: foreground (i.e., normal) or background.[6] An aperiodic task will execute at foreground priority if the sporadic server has not exhausted its execution budget (S_{ss} in Table 1 on page 4). If the budget has been exhausted, the aperiodic task is restricted to background priority. A sporadic server that has been restricted to background priority is not restored to foreground priority or reactivated until its execution budget is replenished.

The execution budget is a nonzero parameter used in the management of the sporadic server. This budget is assigned when the server is created and either decreased or increased over the lifetime of the sporadic server while never exceeding its initial value. The budget is decreased each time the sporadic server handles an event in foreground priority. Further, each time the budget is decreased, a replenishment event is scheduled based on the time the aperiodic event arrived to the sporadic server and the *replenishment period* (T_{ss} in Table 1). The replenishment period is also a nonzero parameter of the sporadic server. The replenishment event, then, is a future point in time when the budget for the sporadic server is scheduled to be increased.

In general, the SSSA can be implemented in an OS's scheduler (e.g., kernel mode) [Shi 01] or within an application (e.g., user mode) [Gonzalez Harbour 91]. When implemented in an OS's kernel, measures of actual CPU execution time used by a process or thread permit more precise accounting and finer manipulation of the sporadic server's execution budget over its lifetime.

The application-level SSSA makes no assumptions about support for sporadic servers in any given OS, lending easy adaptation to a variety of platforms. Comparisons between the appli-

[6] For this report, foreground priority is assumed to be higher than the ceiling of all periodic tasks; likewise, background priority is assumed to be less than the floor of all periodic tasks.

cation-level sporadic server and the full-featured, OS-supported sporadic server show that worst-case performance is the same (except for additional overhead) and the average-case performance can be almost the same when the actual process or thread execution time approaches the worst-case estimation [Gonzalez Harbour 91].

Figure 2 is an example, adapted from Gonzalez Harbour's work [Gonzalez Harbour 91], that depicts the general behavior of an application-level sporadic server.

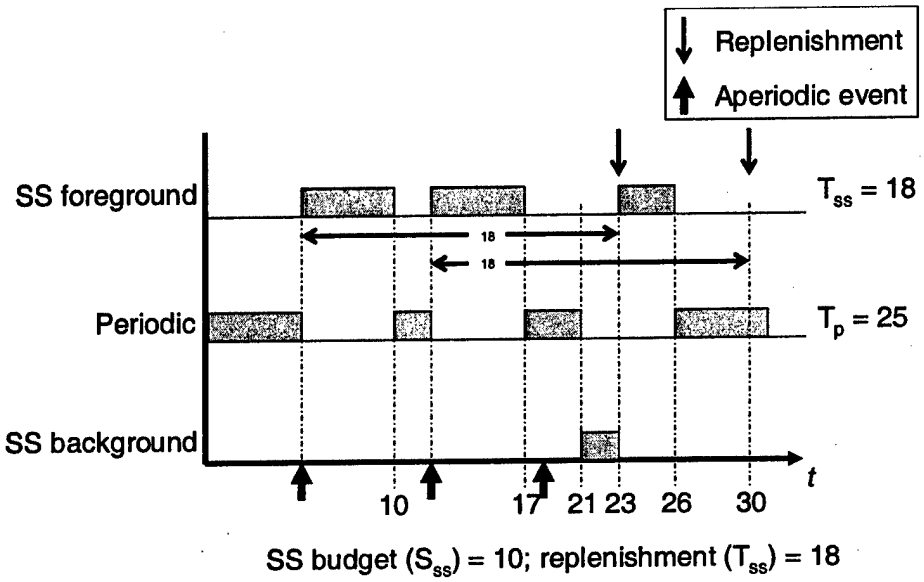


Figure 2: Example of a Sporadic-Server-Controlled Task

In this example, each aperiodic event takes 5 units of time to be serviced. The first two aperiodic requests arrive at t=5 and t=12 and are serviced immediately because, at t=5, the execution budget of the sporadic server is decreased by 5 units of time. That decrease still leaves a remaining execution budget of 5 units that permits the sporadic server to execute at foreground priority. Also at t=5, a replenishment event is scheduled for t=23 (i.e., 23 = event occurring at 5 + replenishment period 18). At t=12, the execution budget is again reduced by 5 units of time, the replenishment is scheduled for t=30, and the sporadic server can still execute at foreground priority. After t=12, the execution budget is exhausted, and when the next aperiodic event arrives at t=18, the sporadic server is restricted to execute at background priority. The additional execution budget for 5 units of time is replenished at the scheduled times of t=23 and t=30, respectively, for the first two requests, thereby restoring the execution budget of the sporadic server.

To implement the SSSA at the application level (i.e., without explicit OS-level support), only two key features of the implementation environment are necessary:

1. some form of synchronous, interprocess, or interthread communication

2. the ability for one process or thread to read and change another process or thread priority

The sporadic server manager, or SSmanager, is a user-level thread that operates at system high priority. The purpose of the SSmanager is to manage one or more sporadic server tasks, or SStasks, each of which processes aperiodic events. An aperiodic task can be converted into an SStask by including two synchronous service requests to the SSmanager: `request()` and `arm()`.

`SSmanager.request()` is a method called by the SStask as soon as the SStask receives an aperiodic event (see Figure 3).

```
// SStask handling aperiodic events
Do while (not done)

  SSmanager.arm (MyThreadID)

  // Wait for aperiodic event request

  OS.wait (event)

  SSmanager.request (MyThreadID, Sa)

  // Do Aperiodic Work

  done = doWork(event);

End Do
```

Figure 3: Pseudocode for SStask

On invocation, `SSmanager.request()` decides whether to permit the SStask to run at foreground priority based on the budget allocated to that SStask and the execution time requested out of that budget (i.e., S_a). If sufficient budget is available, the budget for the SStask is decreased by the requested amount, a replenishment event is scheduled for a later time, and the SStask's priority is set to foreground priority. Otherwise, the request for an execution budget is placed on a pending queue of requests, and the SStask's priority is set to background priority (see Figure 4).

```
void SSmanager::arm(&SSTask) {

  SetPriority (SSTask.ID, SYSTEM.MAX)
}
void SSmanager::request(&SSTask, request_Sa) {

  If SStask.budget >= request_Sa {
    Decrease SStask.budget by request_Sa
    Replenishment (SStask, now()+SStask.Tss, Sa)
    SetPriority (SSTask.ID,  SSTask.foreground)
  }
  Else {
    SavePendingRequest (SSTask, request_Sa)
    SetPriority (SSTask.ID,  SSTask.background)
  }
  End if
}
```

Figure 4: Pseudocode for SSmanager.request() and SSmanager.arm()

`SSmanager.arm()` is also used by the SStask to communicate that the processing of the aperiodic event is complete and that SStask is ready to process another aperiodic request. `SSmanager.arm()` then places the SStask at a system high priority, allowing the latter to wait at a high priority for the aperiodic event. Placing SStask at this priority is necessary (specifically for the application-level SSSA) to allow SStask and SSmanager to acquire and compute the replenishment origin (i.e., `now()+SStask.Tss` in Figure 4) in `SSmanager.request()` based on a time as close as possible to when the aperiodic event arrived at the SStask.

```
SSmanager::replenishment_timer(&SStask, request_Sa) {

   Increase SStask.budget by request_Sa

   If GetPriority(SStask.ID == SStask.background and
      SStask.budget >= GetPendingRequest(SSTask)

      Decrease SStask.budget by GetPendingRequest(SSTask)

      Replenishment (SStask, now()+SStask.Tss,
                     GetPendingRequest(SSTask))

      SetPriority (SSTask.ID,  SSTask.foreground)
   End if
}
```

Figure 5: Pseudocode for SSmanager.replenishment_timer()

Replenishment of the SStask's budget occurs in the SSmanager, usually via an OS-supported timer event. The timer handler simply increases the execution budget for the SStask based on the last honored request for execution time (i.e., S_a). Additionally, the timer handler will check the current priority of the managed SStask. If it's at background priority, the SStask is

still processing, in background, some aperiodic event for a request that could not previously be honored due to the lack of an execution budget.[7] If the previous request can now be honored (i.e., a sufficient budget now exists) and SStask is processing in background, the previous request is honored following the same steps in `SSmanager.request()`.

High-level sequence diagrams covering the sequence of events between the SStask, SSmanager, and the host OS are shown in Figure 6 (for `SSmanager.request()` and `SSmanager.arm()`) and in Figure 7 (for the replenishment timer).

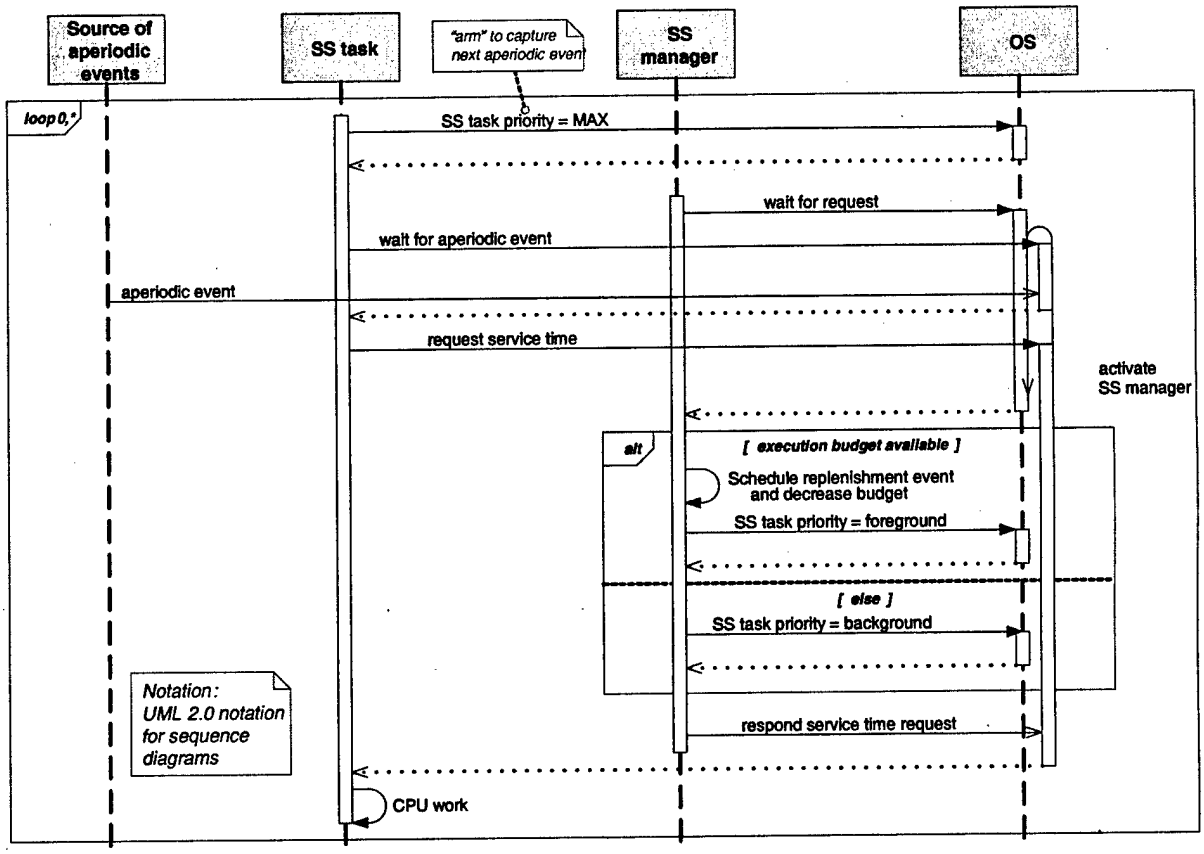


Figure 6: UML 2.0 Sequence Diagram of Application-Level SSSA: Request and Arm

[7] If the SStask is at foreground priority, there is no need to increase its priority. If the SStask is at system high priority, the SStask is armed waiting on an aperiodic event and is not currently processing one.

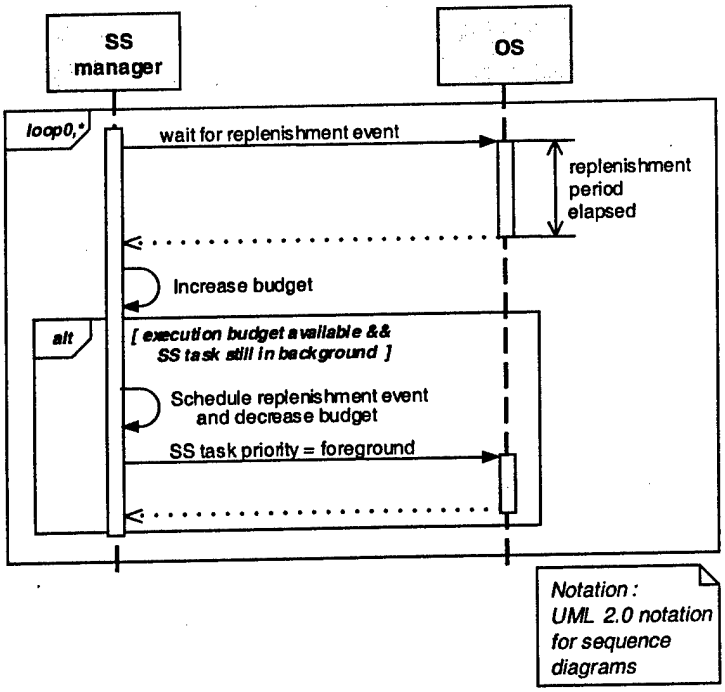


Figure 7: UML 2.0 Sequence Diagram of Application-Level SSSA: Replenishment

4 Reasoning About the Average Latency of Aperiodic Tasks Managed by Application-Level Sporadic Servers

The previous section discussed the SSSA for blending aperiodic and periodic processing in a controlled manner. The goal of this section is to show how queueing theory can be applied to predict the average latency of aperiodic events managed by a sporadic server.

We start by discussing an assembly that has a single aperiodic stream of events with interarrival times governed by an exponential probability distribution and constant service times. Such a situation is known as an M/D/1[8] queueing problem. The aperiodic events are processed under the control of a sporadic server. The assembly also has a single periodic stream of events. The sporadic server executes at the highest priority and allows the aperiodic events to execute whenever the needed budget is available. Otherwise, periodic events execute. Aperiodic events also exploit any available idle time left over from the periodics, that is, background time.

Assumptions. In summary, here are the governing assumptions for this section:

- There is a single stream of aperiodic arrivals. They arrive according to an exponential distribution with mean T_a—that is, if X is a random variable denoting the time interval between successive arrivals, the cumulative probability distribution is

  $$\Pr(X \leq x) = 1 - e^{-\frac{1}{T_a}x}$$

- The execution time is constant, S_a.
- A single application-level sporadic server is used. Its budget is equal to the constant aperiodic service time, $S_{ss}=S_a$. Aperiodics exploit background time, if it is available.
- The assemblies have one or more periodic tasks that run at a lower priority than the sporadic server. (We will focus initially on the special case in which there is a single periodic task with execution time S_p and period T_p.)

[8] *M/D/1* is conventional queueing-theory shorthand denoting queueing systems with an exponential interarrival distribution—M stands for "Markovian"—and a constant execution time—D stands for deterministic.

In this section, our goal is to predict the average latency for the aperiodic events being serviced by a sporadic server.

4.1 Observations About Average Latency When Using a Sporadic Server

One outcome of our work to date is an understanding of which parameters are important for controlling average latency. Looking at Equation (2), it is evident that S_a and T_a (which are implicit in $\rho = S_a/T_a$) are important. The replenishment period (T_{ss}) and the budget (S_{ss}) of the sporadic server are also important. The utilization of the periodic events (U_p) is important, and, perhaps most surprisingly, the period of the periodic events (T_p) is also an important parameter.

To visualize the impact of varying the aforementioned parameters, we ran several simulations and plotted the results shown in Figure 8. The situation being simulated included one aperiodic and one periodic task. The average interarrival interval for the aperiodic is T_a=200; the constant execution time was S_a=10. The period of the periodic task is different for each curve (see the legend in Figure 8). Each curve plots the average latency for the aperiodic events, E[W], as a function of periodic utilization, U_p. For all curves, the budget and replenishment period of the sporadic server are S_{ss}=10 and T_{ss}=100, respectively.

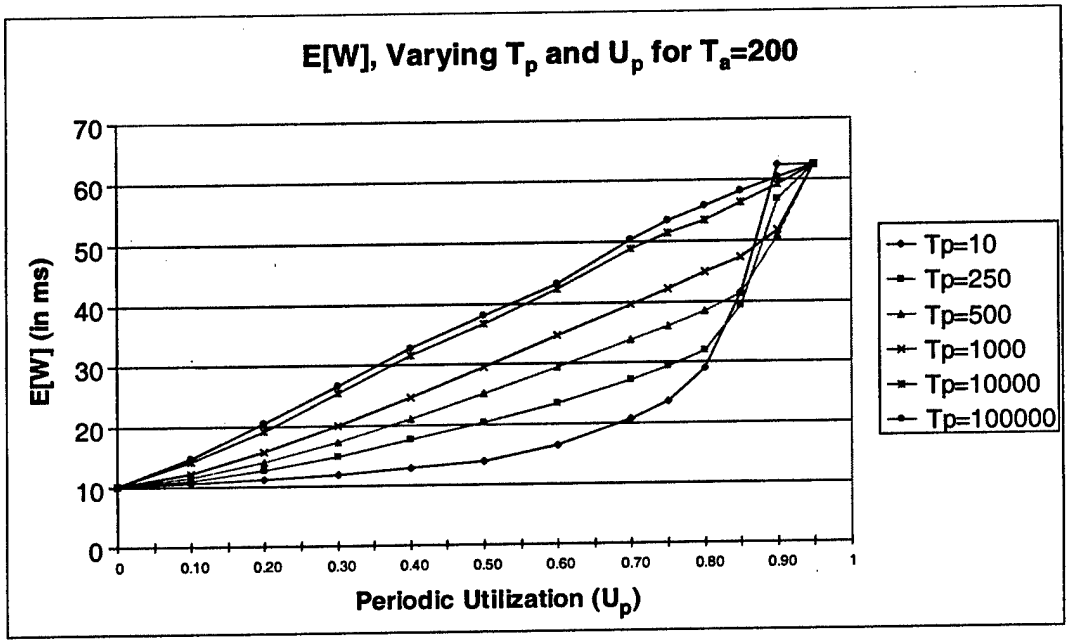


Figure 8: E[W] = f(T_p, U_p) for T_a=200, S_a=S_{ss}=10, and T_{ss}=100

Some observations about the curves in Figure 8 follow:

- Given a period of the periodic (T_p), the average latency of the aperiodic task depends on the periodic utilization. The intuition for that statement is that the opportunity to run in background diminishes as the utilization of the periodic task, U_p, increases.
- The curve for the smallest periodic period looks like a standard queueing curve (see Figure 1). The curve with the largest period looks like it aspires to be a straight line, suggesting that, as the period of the periodic decreases, queueing-theoretic effects are dominating. As the period increases, we are seeing a linear combination of end effects.
- The curves start and eventually reach the same maximum. The intuition for that statement is that when the periodic utilization is 0, the difference in period is immaterial. Furthermore, all the curves eventually reach a point at which there is no longer an opportunity to run in background. This point may be reached at different utilizations for different curves (i.e., different values of T_p).

Next, we explore in more detail the effect of varying these parameters and develop detailed insight for some of the cases and empirically based insight for other cases. First, in Section 4.2, we look at the special case when periodic utilization is zero; effectively, there are no periodics. Then, in Section 4.3, we examine the other extreme where the utilization of the periodics is sufficiently high so that the aperiodic only executes when the needed budget is available from the sporadic server. Finally, in Section 4.4, we examine the case in which the period of the periodic is very small and apply queueing theory to give us insight into the nature of the curves in Figure 8.

4.2 Special Case of No Periodics

When there are no periodics, the CPU is totally available to the aperiodic task. It is exactly the same as a classical queueing problem, so Equation (2) is applicable (which is duplicated below for convenience).

$$E[W] = \left(\frac{\rho}{1-\rho}\right)\left(\frac{E[S_a{}^2]}{2E[S_a]}\right) + E[S_a] \tag{4}$$

Using the example from Figure 8 where $S_a = 10$, $T_a = 200$, and $\rho = S_a/T_a = .05$ and substituting into the formula, we get the following solution:

$$E[W] = \left(\frac{.05}{.95}\right)\left(\frac{100}{20}\right) + 10 = 10.26 \tag{5}$$

The solution above is very close to what is shown in Figure 8—10.25 for $U_p = 0$. For given values of S_a and T_a (which are 10 and 200 in this case), all the curves are "anchored" at this point.

4.3 Special Case of No Background

When the periodic tasks consume enough of the CPU such that there is no background available for the aperiodic events to exploit, all of the aperiodic task's processing must be performed within the budget of the sporadic server. In our example (that is, S_{ss}=10 and T_{ss}=100), only 10 ms[9] of time running at foreground priority is available every 100 ms through the sporadic server.

To help explain further, we provide the following analogy. Imagine customers queueing up to a teller's window in a bank. In the no-periodics case, the teller continuously processes customer requests. In the no-background case, the teller takes care of one customer (recall that the customer request exactly matches the sporadic server's budget of 10 ms) and then does other non-customer work (e.g., paperwork) for 90 ms, while the next customer impatiently waits. From the point of view of customers in the queue, each customer seems to be taking 100 ms. (Let's pretend that customers in the queue cannot distinguish between real customer work and paperwork.) The only saving grace is that customers are pleasantly surprised to find out that once they reach the teller, their request only takes 10 ms. Consequently, from the point of view of customers in the queue, $S_a = 100$; from the point of view of the customer being serviced, $S_a = 10$. To reflect this in the formula, we denote the service time from the queueing perspective as S_q, service time from the server perspective as S_s, and traffic intensity from the queueing perspective as ρ_q. This more general formula is

$$E[W] = \left(\frac{\rho_q}{1-\rho_q}\right)\left(\frac{E[S_q^2]}{2E[S_q]}\right) + E[S_s] \tag{6}$$

Again, using the example from Figure 8 where $S_s = 10$, $S_q = 100$, $T_a = 200$, and $\rho_q = S_q/T_a = .5$ and substituting into the formula, we get the following solution:

$$E[W] = \left(\frac{.5}{.5}\right)\left(\frac{10000}{200}\right) + 10 = 60 \tag{7}$$

The solution above is also very close to what is shown in Figure 8—62.4 ms.

[9] Using milliseconds is arbitrary but a little more concrete than saying "units of time."

4.4 Special Case of Continuous Background

The two previous sections discussed two special cases: U_p=0 and U_p=.9, no periodics and no background, respectively. Our next challenge is to understand the "curves" between these two extremes. We chose to study what we have been calling the "continuous background" case. This case is unrealistic but useful because it reveals much of the problem's queueing-theoretic structure.

When the sporadic server has exhausted its budget, the only way for aperiodics to execute is in background. If S_p=5 and T_p=10, background becomes available in chunks of 5; if S_p=.5 and T_p=1, background becomes available in chunks of .5; if S_p=.05 and T_p=.1, background becomes available in chunks of .05, and so forth. The smaller the period, the more "continuously" background is available. Continuing to reduce S_p and T_p in this manner results in background being very frequently available in infinitesimal quantities—a situation we call *continuous background.* In this case, background processing is equivalent to being continuously processed in a degraded processor. For example, when U_p=.5, background processing in the continuous case is equivalent to executing in a CPU that is half the speed of a full processor. This equivalence to a degraded processor is what makes this an interesting and illuminating special case.[10]

A Sample Timeline. It's helpful to consider a sample timeline to see the different types of time experienced by aperiodic events in this case of continuous background.

Immediately prior to the beginning of the timeline shown in Figure 9, assume that the processor is idle and the sporadic server is loaded with its entire budget of execution time. When the first aperiodic event occurs at t = 100, it is immediately served by the sporadic server. We assume that $S_{ss} = S_a = 20$, $T_{ss} = 145$, U_p=.5 and that T_p is infinitesimally small. Since the sporadic budget is equal to the aperiodic service time (remember we constrain our assemblies to adhere to this restriction), the aperiodic that starts at t=100 completes within the sporadic server's budget of 20 at t=120.

---

[10] On more than one occasion when we described this special case, we were asked, "What about the deadlines of the periodic processes, aren't they being missed?" Or, "If a rate monotonic priority assignment is being used, why does the sporadic server execute at a higher priority?" First, we could pretend that the deadlines are very long and we are using a deadline monotonic priority assignment. However, we are actually ignoring the periodics. The whole point of considering this case is to gain some understanding of the curves in Figure 8. In fact, by treating background processing as a degraded processor, the periodics effectively disappear.

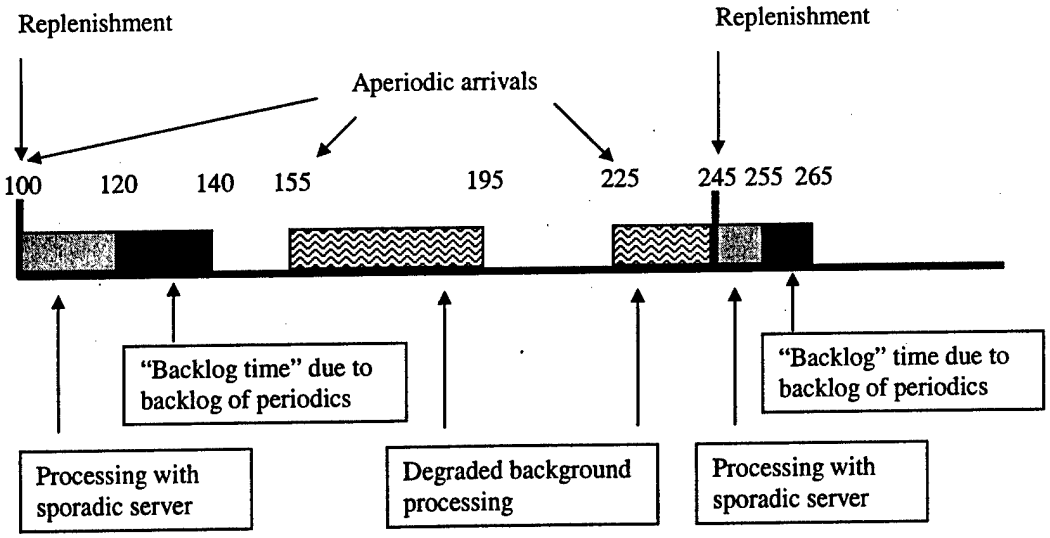


Figure 9: Sample Timeline

"Backlog" Time. While the sporadic server is working, a backlog of undone periodic work is accumulating. At t=120, this backlog is equal to $S_a*U_p = 20*.5 = 10$. However, while continuously working on this backlog of 10 units of periodic work, more undone periodic work accumulates—in this case, 10*.5. This accumulation continues until the work is finished. The following expression is how long it takes from the time the aperiodic event arrives at t=100 until both it and the backlog have completed:

$$S_a + S_a U_p + S_a U_p^2 + S_a U_p^3 + S_a U_p^4 + \cdots = \frac{S_a}{1-U_p} \tag{8}$$

For our sample timeline, $S_a/(1-U_p) = 20/(1-.5) = 40$, which means the backlog interval completes at time t=140.

Equation (8) gives us insight into how to generalize the no-background result. Recall that for the no-background case, $S_q = T_{ss}$. However, the more general result for arbitrary values of U_p is $S_q = S_a/(1-U_p)$. Notice that since $S_a = S_{ss}$, when $U_p = 1-S_{ss}/T_{ss}$ (that is, the periodic utilization is 1 minus the sporadic server utilization), $E[S_q] = T_{ss}$.

Degraded Background Processing. Next, we focus our attention on the block of processing that starts at t=155. That block represents an aperiodic arrival that is completely executed in background. In effect, it is completely executed in a degraded processor. In this example, $U_p = .5$. As a result, the processor is degraded by 50%, and consequently, we would expect the 20 ms service time to take 40 ms. In general, we would expect the S_a service time to take $S_a/(1-U_p)$. Notice that the degraded service time (when executing completely in background) is equal to the service time plus backlog time when executing within a sporadic server.

"Hybrid" Processing. Next, we consider the processing that starts at time t=225 in Figure 9. In this case, service starts in background. Then, when it is partially complete, the sporadic server's budget is replenished, and the remainder of the processing is done in the sporadic server. The case depicted by the timeline shows that half of the service is completed in background, and half is completed in the sporadic server. Note that we are studying the application-level sporadic server, so whatever budget is not used is lost. In our example, five units of time remain to be processed in the sporadic server. Therefore, the five units of remaining capacity are lost.

The background processing requires $(.5*S_a)/(1-U_p)$, and the sporadic server and backlog time also require $(.5*S_a)/(1-U_p)$. As a result, the total is $S_a/(1-U_p)$; this is true regardless of the fraction that completes in background.

4.4.1 Computing Average Queueing Time (E[Q])

Above, we argued that it does not matter whether an aperiodic arrival starts and completes within the sporadic server (i.e., the execution budget was sufficient to process the aperiodic event), gets processed completely in background (i.e., no execution budget is available), or is a hybrid (i.e., the execution budget was replenished prior to completion of the aperiodic event). The customers in the queue always see a delay of $S_a/(1-U_p)$ for each customer served. This fact enables us to compute the average time in the queue, which under a heavy load is usually a major contributor to average latency (see Figure 1 on page 7). To predict the queueing time, we can use Equation (3) as follows:

$$E[Q] = \left(\frac{\rho_q}{1-\rho_q}\right)\left(\frac{E[S_q^2]}{2E[S_q]}\right), \text{ where } S_q=S_a/(1-U_p) \text{ and } \rho_q=S_q/T_a \qquad (9)$$

For the situation in which $T_p=1$ and $T_a=50$, 250, and 500, Figure 10 shows 6 curves;[11] three curves show predicted queueing time and three show actual results from a simulation.

[11] Note that the predicted and actual lines in Figure 10 are so close, their respective curves are difficult to distinguish.

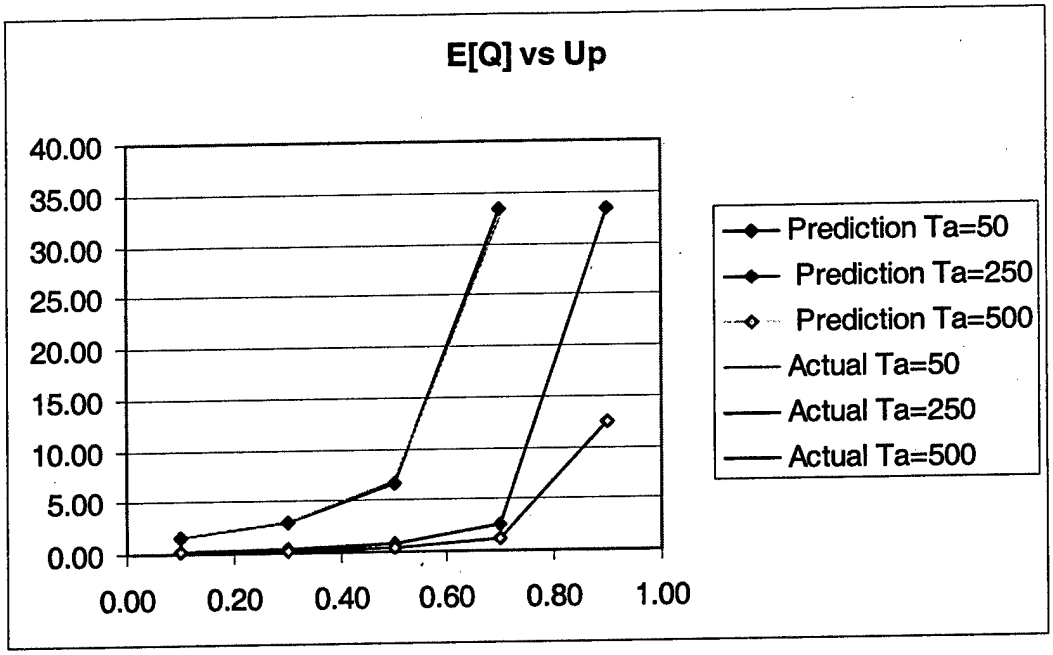


Figure 10: Predicting E[Q]

4.4.2 Computing Average Service Time (E[S_s])

Figure 11 below highlights the actual service times for the timeline shown above in Figure 9. Note that while S_q in Equation (6) is the same regardless of whether an arrival is processed in the sporadic server, in background, or as a hybrid, the actual execution, S_s, varies depending on the situation.

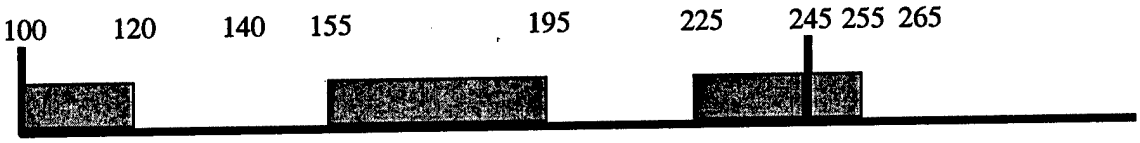


Figure 11: Differing Service Times for the Aperiodic Arrivals

The service time for the arrival serviced at high priority by the sporadic server is S_a. The arrival that is executed completely in background has an execution time of S_s = $\hat{S}_a = S_a/(1-U_p)$. The arrival that executes as a hybrid (where a fraction, α, of its execution time completed in background and the rest with the sporadic server) has an execution of S_s = $\alpha\hat{S}_a + (1-\alpha)S_a$.

These differing service times pose a challenge for deriving an exact formula for predicting average latency. The challenge is to determine the probability associated with each case above.

Overall Approach. The three different situations shown in Figure 11 above can be distinguished by where their busy periods[12] start relative to when replenishment occurs and then by the length of their busy periods. For example, if the following conditions are true, the busy period will consist of one service time of duration S_a (for the first job, which executed within the sporadic server) followed by n-1 service times of length $\hat{S}_a$:

- Replenishment occurs during an idle period.
- Sometime later, a busy period starts.
- The busy period has n jobs.
- The busy period completes before the next replenishment time.

Another busy period might start before the point of replenishment, continue past the point of replenishment, and have a duration less than T_{ss}. In this case, the busy period might comprise n-1 jobs of length $\hat{S}_a$ and one job of length $\alpha\hat{S}_a + (1-\alpha)S_a$ (where $\alpha \leq 1$).

$E[S_s]$ depends on the amount of time from the beginning of the busy period until the point of replenishment (or what we call the time to replenishment) and on the length of the busy period. Because of that, our approach is to determine the distribution function describing the time to replenishment for the start of a busy period and use that value in conjunction with the distribution function for the length of the busy period as the basis for computing $E[S_s]$.

Let Tr be the random variable denoting the time to replenishment at the beginning of busy periods. Our goal is to compute

$$E[S_s] = \int_{[0,Tss]} E[S_s \mid Tr = t] dF_{Tr}(t) \tag{10}$$

where $E[S_s \mid Tr=t]$ is the conditional expectation of S_s, given that the time to replenishment is Tr=t and $F_{Tr}(t)$ is the cumulative distribution function (CDF) for the time to replenishment. Given that Tr can be neither negative nor greater than the replenishment period, t can only take values in the interval $[0, T_{ss}]$. The CDF is the probability that Tr is less than or equal to t (represented as $Pr(Tr \leq t)$).

To gain insight into the nature of the distribution of Tr, we ran several simulations and plotted the histogram shown in Figure 12 for one of them. The parameters for this simulation were $S_{ss}=10$, $T_{ss}=100$, $U_p=.60$, $S_a=10$, $T_a=200$, and $T_p=1$.

[12] A *busy period* is a continuous interval of time during which the server (or processor) is busy.

Each bar except the first bar represents the average probability density over the interval calculated by taking the proportion of samples falling within the interval divided by the length of the interval. The first bar is the proportion of samples whose busy period starts at Tr=0.

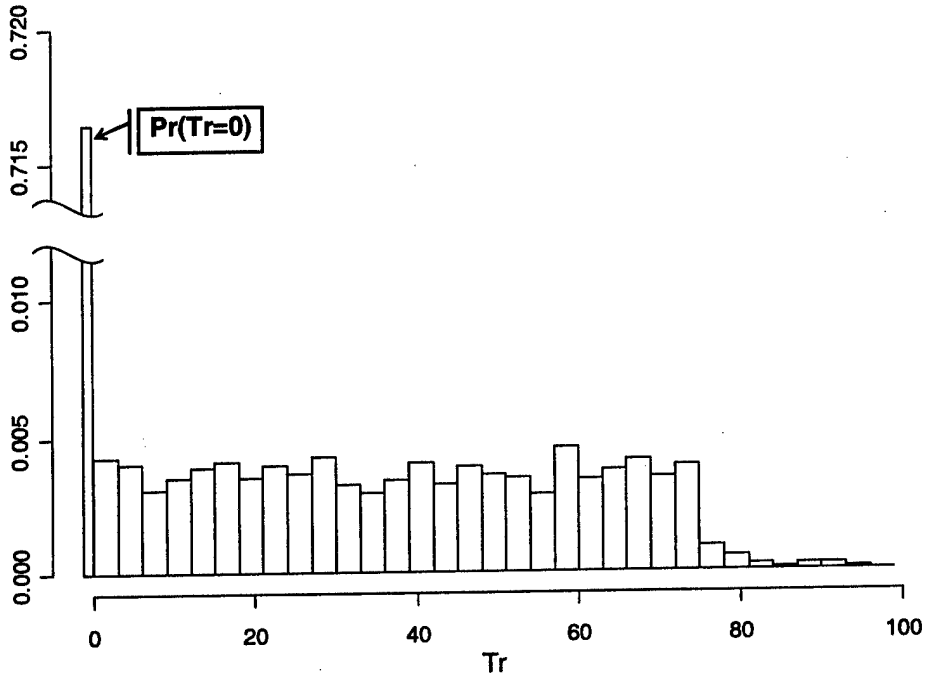


Figure 12: Histogram of Tr

Notice that Tr has what appears to be a uniform density between Tr>0 and approximately 75, has a much higher density (approximately 0.716) at Tr=0, and has a density that tails off from 75 to 100. Therefore, it appears that $f_{Tr}(t)$, the density function of time to replenishment, has the form

$$f_{Tr}(t) = \begin{cases} au_0(t) \ for\ t = 0 \\ \dfrac{1}{T_{ss} - h} \ for\ t\ in\ (0, T_{ss} - h) \\ d(t) \quad for\ t\ in\ [T_{ss} - h, T_{ss}] \end{cases} \tag{11}$$

where a is a constant and $u_0(t)$ is the Dirac delta function [Kleinrock 75]. This result motivated us to break Equation (10) into two terms: one for Tr=0 and one for Tr>0 as shown in Equation (12) below.

$$E[S_s] = E[S_s \mid Tr = 0]\Pr(Tr = 0) + \int_{(0,Tss]} E[S_s \mid Tr = t] f_{Tr}(t)dt \tag{12}$$

Evaluating Pr(Tr=0). To understand how to compute Pr(Tr=0), consider Figure 13 below. The x-axis is time, and the y-axis is time to replenishment (Tr). Time to replenishment starts

at Tr=0 and then stays at that level until a busy period starts due to an aperiodic arrival. At that moment, the sporadic server's budget is consumed, and the next replenishment is scheduled to occur one replenishment period later, making $Tr=T_{ss}$. Tr then decreases at a rate of one until it reaches zero, and then replenishment occurs. Replenishment can occur either during a busy period—a dark region on the bottom timeline—or during an idle period. If it occurs during a busy period, Tr is once again immediately set equal to T_{ss} and starts to decrease. If replenishment occurs during an idle period, Tr remains at zero until the next arrival occurs. When that happens, Tr is set equal to T_{ss} and starts decreasing.

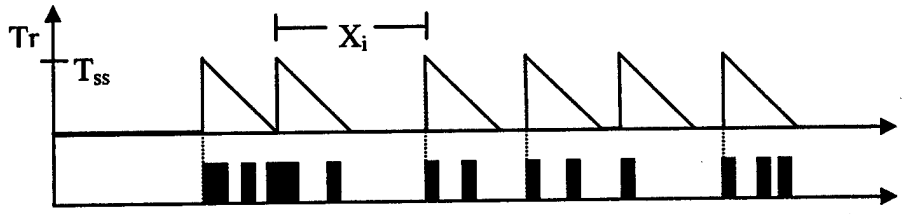


Figure 13: Time to Replenishment and Busy Periods

The question is, how do we characterize the proportion of time in which the system is in the Tr=0 state? First, note that several busy periods can occur during the period of time from $Tr=T_{ss}$ to Tr=0. In fact, our current analysis depends on the replenishment period being large enough for several busy-idle cycles to occur during one replenishment period. However, at most, one of those busy periods can be preceded by an interval in which Tr=0. And such an interval (of Tr=0) only occurs when the previous replenishment occurs during the idle interval that immediately precedes the busy period.

To compute Pr(Tr=0), we will associate one busy period with each time a replenishment occurs. If replenishment occurs during an idle period, we will associate it with the first busy period after the replenishment. If the replenishment occurs during a busy period, that period will be associated with it (see Figure 13 above). Therefore the interval denoted by X_i in Figure 13 always begins during (or at the beginning of) a busy period associated with a replenishment.

Tr=0 occurs when a busy period has a replenishment associated with it **and** when the replenishment occurs during an idle interval. Assuming these two events are independent, $Pr(Tr=0) = p_I * p_R$ where p_R = Pr(the busy period is associated with a replenishment) and p_I = Pr(replenishment occurs during an idle period). To calculate p_I and p_R, we will use some results from renewal theory.

A Result from Renewal Theory. A renewal process is defined as a stochastic process that counts the number of arrivals of events, N(t), that occur in the interval [0,t] where the time between arrivals is determined by a sequence of nonnegative, independent, identically distributed random variables $\{X_i, i=1, 2, \ldots\}$. X_i is the time between arrivals i-1 and i [Ross 96]. N(t) can then be defined in terms of X_i.

$$N(t) = \max\{n \mid \sum_{i=1}^{n} X_i \leq t\} \tag{13}$$

An alternating renewal process is defined as a sequence of random vectors (Z_i, Y_i) where Z_1 occurs, followed by Y_1, followed by Z_2, followed by Y_2, and so on. You can think of a system as being in one of two states: it is in state "on" for an interval of length Z_1, followed by state "off" for an interval of lengthY_1, and so forth. The Z_is are independent and identically distributed, as are the Y_is. We will use the following theorem from Ross where $P_{on}(t)$ is the probability that the system is "on" at time t [Ross 96].

Theorem: If $P_{on}(t)$ = Pr(system is on at time t) and $E[Z_i+Y_i]$ is finite (and the distribution of Z_i+Y_i is nonlattice[13]) then

$$\lim_{t\to\infty} P_{on}(t) = \frac{E[Z_i]}{E[Y_i]+E[Z_i]} \tag{14}$$

This theorem says that the limiting probability the system will be "on" is equal to the proportion of time that it is "on" during an average on-off cycle.

Calculating p_I. We can view an on-off cycle as an idle interval, I_i, followed by a busy period, B_i. If we consider the idle interval as "on time" and use the above theorem, we get the following equation:

$$p_I = \frac{E[I_i]}{E[B_i]+E[I_i]} \tag{15}$$

Calculating p_R. Recall that p_R is the probability that a busy period is associated with replenishment. We can define a renewal as the time at which the sporadic server's budget is consumed and X_i (Figure 13) as the time between such renewals. To compute p_R, we must determine the average number of cycles that can occur during an interval between renewals, where a cycle is C_i=B_i+I_i:

$$p_R = \frac{E[B_i]+E[I_i]}{E[X_i]} \tag{16}$$

If the cycle associated with replenishment is thought of as an "on interval" and the other cycles during X_i are thought of as an "off interval," Equation (16) can be viewed as another application of Equation (14). Therefore, to determine p_R, we first need to determine $E[X_i]$.

---

[13] A nonnegative random variable is said to be *lattice* if it takes on values only at points that are multiples of some nonnegative number, d [Ross 96].

Calculating E[X_i]. Note two things:

1. X_i is equal to T_{ss} if the i^{th} replenishment occurs during a busy period.
2. X_i is equal to T_{ss} plus the remaining portion of an already started idle period, if the i^{th} replenishment occurs during an idle period.

Therefore, we need to understand the distribution of the remaining portion of idle time to determine the distribution for X_i. I denotes idle time. Since we assume that the arrival distribution is exponential and the exponential distribution is memoryless,[14] the idle time distribution must also be exponential [Kleinrock 75].

Let $I_R(t)$ denote the remaining idle time at time t. This time is known as the residual time or the forward recurrence time.

The limiting distribution for $I_R(t)$ is

$$\Pr(I_R \le x) = \int_0^x \overline{F}_I(\xi)d\xi / E[I] \tag{17}$$

where $\overline{F}_I(x) = 1\text{-}F_I(x)$ [Ross 96]. Since I is exponentially distributed with mean T_a, the "memoryless" property of an exponential distribution would suggest that I_R is also exponentially distributed with the same mean. Using Equation (17) to compute the distribution of I_R confirms this.

Using the distribution of I_R and letting $p_B = 1\text{-}p_I$ we can compute the distribution for X_i as follows:

$$\Pr(X_i \le x) = \Pr(X_i \le x \mid Busy)p_B + \Pr(X_i \le x \mid Idle)p_I \quad \text{[15]} \qquad \text{for } x \ge T_{ss}$$

$$\Pr(X_i \le x) = 0 \qquad \text{for } x < T_{ss}$$

Since we know that X_i is T_{ss} when replenishment occurs during a busy period, the first term is simply $1*p_B$. The second term represents the case in which replenishment occurs during an idle interval and comprises T_{ss} plus some remaining idle time. Therefore

$$\Pr(X_i \le x) = p_B + \Pr(T_{ss} + I_R \le x)p_I \qquad \text{for } x \ge T_{ss} \tag{18}$$

Figure 14 below shows the shape of the cumulative distribution function, $F_{Xi}(t)$, for X_i.

---

[14] *Memoryless* means that the amount of idleness that has already occurred does not affect how much idleness is left for any given idle interval.

[15] *Busy* means replenishment during a busy period, and *idle* means replenishment during an idle period.

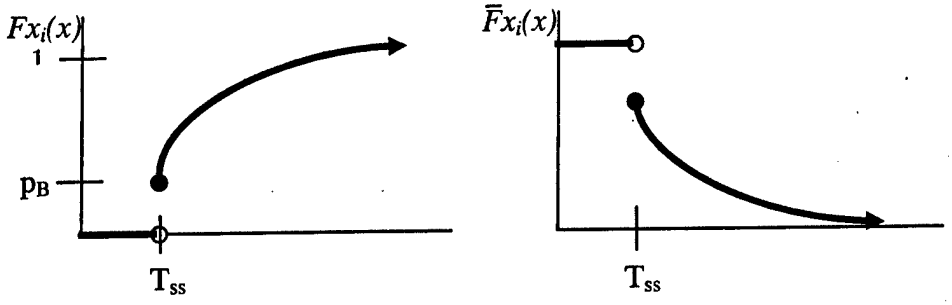


Figure 14: CDF for X_i

The expectation of X_i can be calculated by conditioning on whether X_i occurs in an idle or busy period, that is

$$E[X_i] = E[X_i \mid Busy]p_B + E[X_i \mid Idle]p_I = T_{ss}p_B + (T_{ss} + E[I_R])p_I = T_{ss} + E[I_R]p_I \quad (19)$$

Substituting the expression for $E[X_i]$ from Equation (19) into Equation (16) results in this equation:

$$p_R = \frac{E[B_i] + E[I_i]}{T_{ss} + E[I_R]p_I} \quad (20)$$

which is equal to the probability that a cycle is associated with a replenishment. Since $Pr(Tr=0) = p_I * p_R$, the following equation applies:

$$\Pr(Tr = 0) = \frac{p_I(E[B_i] + E[I_i])}{T_{ss} + E[I_R]p_I} \quad (21)$$

Evaluating $E[S_s \mid Tr=0]$. Just as a reminder, our goal is to use Equation (12) to compute $E[S_s]$. We are still calculating the first term of Equation (12). We have worked out $Pr(Tr=0)$, and now we turn our attention to $E[S_s \mid Tr=0]$.

To compute $E[S_s \mid Tr=0]$, we condition on the number of arrivals in a busy period. In our case, the distribution for the number of arrivals, BP, in an M/D/1 busy period[16] is the following [Kleinrock 75]:

$$\Pr(BP = n) = \frac{(n\rho_q)^{n-1}}{n!} e^{-n\rho_q} \text{ where } \rho_q = \frac{\hat{S}_a}{T_a} \quad (22)$$

[16] We are justified in using this since the previous section argued that, from a queueing perspective, S_q is equal to $S_s/(1-U_p)$ regardless of whether it executes within a sporadic server, in background, or as a hybrid.

When Tr=0, the busy period always starts with an arrival that is "greeted by" a fully loaded sporadic server. This first arrival will have a service time of S_a. Depending on how large T_{ss} is relative to $\hat{S}_a$, the busy period continues with some number of additional arrivals that execute in background and consequently have service times of $\hat{S}_a$. If the busy period is long enough, it might contain one or more "hybrid" service times as well.

For now, we will ignore the hybrid services (in which case, our estimate will be on the high side). Later, we will show an algorithm that accounts for the hybrid case. The following expression is a pessimistic approximation of the duration of a busy period of length i, given that Tr=0:

$$S_a + (i-1)\hat{S}_a \tag{23}$$

This equation is pessimistic since it approximates a hybrid's execution—which is $\alpha\hat{S}_a + (1-\alpha)S_a$—using simply $\hat{S}_a$. We use this approximation to compute an approximation for $E[S_s|\ Tr=0]$.

Given a very large number of busy periods (denoted as N) and the strong law of large numbers [Ross 96], there are approximately Pr(BP=i)*N busy periods of length i [Cinlar 97]. Equation (23) expresses that for each busy period of length i, there is one arrival with a service time of S_a and i-1 with service times of $\hat{S}_a$. Therefore

$$E[S_s \mid T_r = 0] = \frac{\sum_{i=1}^{\infty} \Pr(BP = i) N[S_a + (i-1)\hat{S}_a]}{\sum_{i=1}^{\infty} \Pr(BP = i)(N)(i)}$$

which reduces to

$$E[S_s \mid T_r = 0] = \frac{\sum_{i=1}^{\infty} \Pr(BP = i)[S_a + (i-1)\hat{S}_a]}{E[BP]} \tag{24}$$

where Pr(BP=i) is defined in Equation (22) and $E[BP]=1/(1-\rho)$.

Therefore, an approximation for the first term in Equation (12) is

$$E[S_s \mid Tr = 0]\Pr(Tr = 0) = \left(\frac{p_I(E[B_i] + E[I_i])}{T_{ss} + E[I_R]p_I}\right)\frac{\sum_{i=1}^{\infty} \Pr(BP = i)[S_a + (i-1)\hat{S}_a]}{E[BP]} \tag{25}$$

Calculating $\int_{(0,Tss]} E[S_s \mid Tr = t] f_{Tr}(t)dt$. Our next step is to consider the case in which Tr is greater than zero. To do this, we need to determine the cumulative distribution function for Tr, $F_{Tr}(t)$.

Recall that Tr is a random variable denoting the time to replenishment for the beginning of a busy period. To compute $\Pr(Tr \le t)$, we condition on whether Tr equals zero or is greater than zero. Recall that

$$\Pr(Tr \le t) = \Pr(Tr \le t \mid Tr = 0)\Pr(Tr = 0) + \Pr(Tr \le t \mid Tr > 0)(1 - \Pr(Tr = 0)) \tag{26}$$

The first term reduces to Pr(Tr=0); Equation (21) addresses this case.

Tr>0 occurs when X_A,[17] the time since the last renewal (known as the age or the backward recurrence time), is less than T_{ss}. Therefore, $\Pr(Tr \le t \mid Tr > 0)$ is equivalent to $\Pr(Tr \le t \mid X_A < T_{ss})$, however

$$\Pr(Tr \le t \mid X_A < T_{ss}) = \Pr(T_{ss} - X_A \le t \mid X_A < T_{ss}) = 1 - \Pr(X_A < T_{ss} - t \mid X_A < T_{ss}) \tag{27}$$

Since the limiting distributions of the forward and backward recurrence times of a renewal process are the same, we can use Equation (17) to compute $\Pr(X_A \le t)$ [Ross 96]:

$$\Pr(X_A \le t) = \int_0^t \overline{F}_X(\xi)d\xi / E[X] \tag{28}$$

Using Equation (28) and conditional probability and referring to Figure 14

$$\Pr(X_A < t \mid X_A < T_{ss}) = \frac{\int_0^t \overline{F}_X(\xi)d\xi}{\int_0^{T_{ss}} \overline{F}_X(\xi)d\xi} = \frac{\int_0^t 1d\xi}{\int_0^{T_{ss}} 1d\xi} = \frac{t}{T_{ss}} \tag{29}$$

Therefore

$$\Pr(Tr \le t \mid Tr > 0) = 1 - \Pr(X_A < T_{ss} - t \mid X_A < T_{ss}) = 1 - \frac{T_{ss} - t}{T_{ss}} = \frac{t}{T_{ss}}$$

which is a uniform distribution over [0, T_{ss}]. The cumulative distribution function for Tr is

[17] The letter "A" in X_A stands for "age." If one considers a renewal to be a birth, X_A represents the time since the last renewal. This should not be confused with X_i, which represents the time between successive renewals.

$$\Pr(Tr \le t) = \Pr(Tr = 0) + \frac{t}{T_{ss}}(1 - \Pr(Tr = 0))$$

and the density function for Tr is

$$f_{Tr}(t) = \frac{dF_{Tr}(t)}{dt} = \Pr(Tr = 0)u_0(t) + \frac{1 - \Pr(Tr = 0)}{T_{ss}}$$

which is similar in form to Equation (11).

This results in

$$\int_{(0,Tss]} E[S_s \mid Tr = t] f_{Tr}(t)dt = \int_{(0,Tss]} E[S_s \mid Tr = t] \frac{1 - \Pr(Tr = 0)}{T_{ss}} dt \tag{30}$$

Accounting for "blackout." Figure 12 showed that Tr is not actually uniformly distributed over the entire interval [0, T_{ss}] but instead is uniformly distributed over the shorter interval [0, T_{ss}-h]. This distribution makes sense since each renewal occurs either during or at the beginning of a busy period. By definition, the next busy period cannot begin until the current one ends, resulting in a "blackout" period, H, for Tr from T_{ss}-H to T_{ss}. Let H be a random variable denoting the duration of this blackout period.

Assume that H=h. Due to this blackout period, $\Pr(Tr \le t \mid Tr > 0)$ is not actually equivalent to $\Pr(Tr \le t \mid X_A < T_{ss})$. Rather, it is equivalent to $\Pr(Tr \le t \mid h \le X_A < T_{ss})$. This difference changes Equation (29) to

$$\Pr(X_A < t \mid h \le X_A < T_{ss}) = \frac{\int_0^t \overline{F}_X(\xi)d\xi}{\int_h^{T_{ss}} \overline{F}_X(\xi)d\xi} = \frac{\int_0^t 1 d\xi}{\int_h^{T_{ss}} 1 d\xi} = \frac{t}{T_{ss} - h} \tag{31}$$

and Equation (30) to

$$\int_{(0,Tss]} E[S_s \mid Tr = t] f_{Tr}(t \mid h)dt = \int_{(0,Tss-h]} E[S_s \mid Tr = t] \frac{1 - \Pr(Tr = 0)}{T_{ss} - h} dt$$

For H=h, $f_{Tr}(t \mid h)$ has the form

$$f_{Tr}(t \mid h) = \begin{cases} au_0(t) \ for\ t = 0 \\ \dfrac{1}{T_{ss} - h} \quad for\ t\ in\ (0, T_{ss} - h) \\ 0 \quad for\ t\ in\ [T_{ss} - h, T_{ss}] \end{cases}$$

where d(t) in Equation (11) is actually 0 for any given H=h. Now we must account for the fact that H is a random variable; therefore, we must determine $f_{Tr}(t)$:

$$f_{Tr}(t) = \Pr(Tr = 0)u_0(t) + (1 - \Pr(Tr = 0)) \int_{(0,Tss]} f_{Tr}(t \mid h) f_H(h) dh$$

Letting P=Pr(Tr=0) and Q=1-P, we have

$$f_{Tr}(t) = Pu_0(t) + Q \int_{(0,Tss]} \frac{1}{T_{ss} - h} f_H(h) dh \tag{32}$$

where $f_H(h)$ is the density function for H. To determine the distribution function for H, we condition on whether the first busy period in the renewal interval is one that starts within the sporadic server (*SS*) or is a hybrid (*hybrid*). Note that

$$f_H(h) = f(h \mid SS) \Pr(SS) + f(h \mid hybrid) \Pr(hybrid) \tag{33}$$

where Pr(SS)=p_I and Pr(hybrid)=1-p_I.

First, we will focus on $f(h \mid SS)$. In this case, replenishment has occurred in the idle interval before the busy period starts. The busy period (and hence the blackout period, H) starts when the next arrival occurs. H can take on only a finite set of values.[18] When the duration of the busy period is less than or equal to T_{ss}, $\Pr(H = \hat{S}_a \mid SS) = \Pr(BP = 1)$, $\Pr(H = 2\hat{S}_a \mid SS) = \Pr(BP = 2)$, and so forth. However, when the duration of the busy period is greater than T_{ss}, H can take on other values—in general, those in the following set:

$$\left\{ j\hat{S}_a \bmod T_{ss} \mid j = 1, \frac{LCM(\hat{S}_a, T_{ss})}{\hat{S}_a} \right\}$$

For example, if $\hat{S}_a$=30 and T_{ss}=100, H can be equal to 30, 60, 90, 20, 50, 80, 10, 40, 70, and 0. The probability function for this can be expressed as

[18] Assuming $\hat{S}_a$ is rational.

$$\Pr(H = j\hat{S}_a \bmod T_{ss} \mid SS) = \Pr(BP = i*L + j)$$

$$for\ L = \frac{LCM(\hat{S}_a, T_{ss})}{\hat{S}_a},\ j = 1,\ldots, L\ and\ i \geq 0$$

and therefore

$$f_H(h \mid SS) = \sum_{j=1}^{L} u_0(h - j\hat{S}_a \bmod T_{ss}) \sum_{i=0}^{\infty} \Pr(BP = i*L + j) \quad [19] \qquad (34)$$

Now, we will focus on $f(h \mid hybrid)$. In this case, replenishment occurs during a busy period of which the blackout period is a subinterval. When the busy period is less than T_{ss} in duration, we assume that replenishment is equally likely to occur at any time during that busy period. Therefore, the time from replenishment to the end of the busy period is uniformly distributed over the length of the busy period, that is

$$f_H(h \mid Hybrid, BP = i, i\hat{S}_a < T_{ss}) = \frac{1}{i\hat{S}_a}\ for\ h \in [0, i\hat{S}_a]$$
$$= 0\ \ otherwise$$

When the length of the busy period exceeds T_{ss}, the first replenishment must be within T_{ss} of the busy period's beginning. If the first replenishment is exactly at the beginning of the busy period, the blackout period is $m = i\hat{S}_a \bmod T_{ss}$. As the replenishment moves away from the beginning of the busy period, h decreases until it is 0, jumps to T_{ss}, and then decreases until its value becomes m again when the replenishment occurs at T_{ss}. Therefore, when the busy period is greater than or equal to T_{ss}, the probability density function of its length, H, is given by

$$f_H(h \mid Hybrid, i\hat{S}_a \geq T_{ss}) = \frac{1}{T_{ss}}\ for\ h \in [0, T_{ss}]$$
$$= 0\ \ otherwise$$

Therefore

$$f_H(h \mid Hybrid) = \sum_{i=1}^{\infty} G_i(h) \Pr(BP = i),\ where$$

$$G_i(h) = \frac{1}{\min(i\hat{S}_a, T_{ss})}\ for\ h \in [0, \min(i\hat{S}_a, T_{ss})] \qquad (35)$$
$$= 0 \qquad\qquad otherwise$$

[19] Recall that $u_0(x)$ is the Dirac delta function. Therefore, $u_0(x-a)$ has a nonzero (actually infinite) value when x=a and is zero everywhere else.

Note that we have made the assumption that the distribution governing BP does not depend on whether the busy period starts within a sporadic server or starts with a hybrid.

Combining Equations (33), (34), and (35), we have

$$f_H(h) = \Pr(SS)\sum_{j=1}^{L} u_0(h - j\hat{S}_a \bmod T_{ss})\sum_{i=0}^{\infty}\Pr(BP = i*L + j) + \Pr(hybrid)\sum_{i=1}^{\infty} G_i(h)\Pr(BP = i) \tag{36}$$

Combining Equations (32) and (36), we have

$$f_{Tr}(t) = Pu_0(t) + Q\ \Pr(SS)\sum_{j=1}^{L} M_j(t)\sum_{i=0}^{\infty}\Pr(BP = i*L + j) + Q\Pr(Hybrid)\int_{(0,Tss]}\frac{1}{T_{ss} - h}\sum_{i=1}^{\infty} G_i(h)\Pr(BP = i)dh \tag{37}$$

Integrating Equation (37), we have

$$f_{Tr}(t) = Pu_0(t) + Q\ \Pr(SS)\sum_{j=1}^{L} M_j(t)\sum_{i=0}^{\infty}\Pr(BP = i*L + j) + Q\Pr(Hybrid)\sum_{i=1}^{\left\lfloor \frac{Tss}{\hat{S}a} \right\rfloor}\Pr(BP = i)I_i(t) + Q\Pr(Hybrid)\sum_{i=\left\lfloor \frac{Tss}{\hat{S}a} \right\rfloor + 1}^{\infty}\Pr(BP = i)\frac{1}{T_{ss}}\ln\left(\frac{T_{ss}}{t}\right) \tag{38}$$

where

$$I_i(t) = \begin{cases} \dfrac{1}{i\hat{S}_a}\ln\left(\dfrac{T_{ss}}{T_{ss} - i\hat{S}_a}\right), & for\ t < T_{ss} - i\hat{S}_a \\ \dfrac{1}{i\hat{S}_a}\ln\left(\dfrac{T_{ss}}{t}\right), & for\ t \geq T_{ss} - i\hat{S}_a \end{cases}$$

and

$$M_j(t) = \begin{cases} \dfrac{1}{T_{ss} - j\hat{S}_a \bmod T_{ss}}, & for\ t \le T_{ss} - j\hat{S}_a \bmod T_{ss} \\ 0, & for\ t > T_{ss} - j\hat{S}_a \bmod T_{ss} \end{cases}$$

Now that we have $f_{Tr}(t)$, we are ready to evaluate the right-hand side of Equation (10):

$$E[S_s] = \int_{[0,Tss]} E[S_s \mid Tr = t] dF_{Tr}(t) =$$

$$P\, E[S_s \mid Tr = 0] +$$

$$Q \Pr(SS) \sum_{j=1}^{L} \left(\frac{1}{T_{ss} - j\hat{S}_a \bmod T_{ss}} \right) \sum_{i=0}^{\infty} \Pr(BP = i * L + j) \int_0^{T_{ss} - j\hat{S}_a \bmod T_{ss}} E[S_s \mid Tr = t] dt +$$

$$Q \Pr(Hybrid) \left[\sum_{i=1}^{\left\lfloor \frac{Tss}{\hat{S}a} \right\rfloor} \Pr(BP = i) \left[\frac{1}{i\hat{S}_a} \ln\left(\frac{T_{ss}}{T_{ss} - i\hat{S}_a} \right) \int_0^{T_{ss} - i\hat{S}_a} E[S_s \mid Tr = t] dt + \right. \right.$$

$$\left. \frac{1}{i\hat{S}_a} \int_{T_{ss} - i\hat{S}_a}^{T_{ss}} \ln\left(\frac{T_{ss}}{t} \right) E[S_s \mid Tr = t] dt \right] +$$

$$\left. \sum_{i=\left\lfloor \frac{Tss}{\hat{S}a} \right\rfloor + 1}^{\infty} \Pr(BP = i) \frac{1}{T_{ss}} \int_0^{T_{ss}} \ln\left(\frac{T_{ss}}{t} \right) E[S_s \mid Tr = t] dt \right]$$

Evaluating $E[S_s \mid Tr=t]$. To complete our analysis, we need an algorithm to compute E[Ss| Tr=t]. Recall that we offered a pessimistic estimate above. A more general form of Equation (24) is given by

$$E[S_s \mid T_r = t] = \frac{\sum_{i=1}^{\infty} \Pr(BP = i) V_i(t)}{E[BP]}$$

where $V_i(t)$ is the total duration of i service times, starting at Tr=t. Calculating $V_i(t)$ requires calculating how many hybrids will occur during the busy period and what the service time is for each of those hybrids.

The number of hybrids in the busy period is given by

$$k_i(t) = \left\lfloor \frac{i\hat{S}_a + T_{ss} - t}{T_{ss}} \right\rfloor + \left\lceil \frac{(i\hat{S}_a + T_{ss} - t) \bmod T_{ss}}{T_{ss}} \right\rceil - 1$$

A hybrid has one part that occurs before replenishment and one that occurs after it. The duration of the part that occurs before replenishment for the j^{th} hybrid is

$$d_j(t) = \left(t + (j-1)T_{ss}\right) - \left\lfloor \frac{t + (j-1)T_{ss}}{\hat{S}_a} \right\rfloor \hat{S}_a$$

The total duration of the hybrid is

$$d_j(t) + S_a - d_j(t)(1-U_p) = S_a + d_j(t)U_p = \hat{S}_a(1-U_p) + d_j(t)U_p$$

The duration of the busy period is

$$V_i(t) = \begin{cases} (i-k_i(t))\hat{S}_a + \sum_{j=1}^{k_i(t)} \left(\hat{S}_a(1-U_p) + d_j(t)U_p\right) = \left(i - k_i(t)U_p\right)\hat{S}_a + \sum_{j=1}^{k_i(t)} d_j(t)U_p & \text{for } t>0 \\ S_a + (i-1-k_i(T_{ss})U_p)\hat{S}_a + \sum_{j=1}^{k_i(T_{ss})} d_j(T_{ss})U_p & \text{for } t=0 \end{cases}$$

4.4.3 Areas of Ongoing Work

In Section 4.4.1, we made several simplifying assumptions that we are in the process of relaxing:

- In computing Pr(Tr=0), we assumed there would be many busy-idle cycles over the course of a replenishment period. In other words, we assumed that $(E[B]+E[I])/T_{ss} < 1$. When T_{ss} is small or the traffic intensity is high enough to cause E[B] to exceed T_{ss}, this assumption is violated. Such a violation can cause inaccuracy in our prediction of Pr(Tr=0) and, in turn, an inaccurate weighted average of $E[S_s|\ Tr=0]$ and $E[S_s|\ Tr>0]$.
- When computing the blackout time for the hybrid case, we assumed that the first replenishment of a busy period is equally likely to occur anywhere in $[0,\ \min(i\hat{S}_a, T_{ss})]$. However, the length of the blackout period might be constrained after a busy period occurs. For example, when one busy period ends very close to replenishment and the next one starts shortly thereafter, there is a constraint on how short the blackout period can be. Figure 15 shows an example of that situation. The busy period BP_1 was hit by a replenishment at time 460, creating a blackout h_1=40 for when the following busy period could start. We can see that the blackout h_2 created by BP_2 is 30, but even if BP_2 started right after BP_1 finished at time 500, the blackout h_2 could be no less than 20.
- In computing $E[S_s|\ Tr=0]$ and $E[S_s|\ Tr=t]$, when t>0 we condition on the number of arrivals in the busy period and then uncondition using the M/D/1 busy period distribution (see Equation (22)). However, due to an effect known in renewal theory as length biasing [Ross 96], there is a bias toward replenishments occurring in relatively long busy periods.

In other words, the distribution of the lengths of hybrid busy periods is different from the distribution of all the busy periods; the probability is shifted toward longer busy periods.

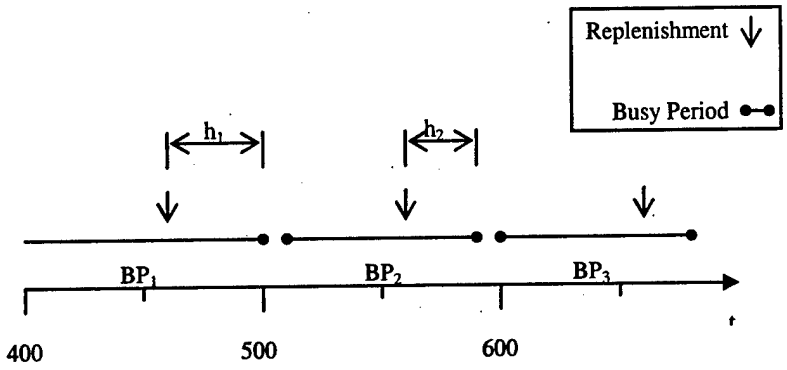


Figure 15: Tr Blackout Dependency on Previous Blackout

We are currently looking into generalizing the theory to account for the cases in which our assumptions are not appropriate.

4.4.4 Empirical Evidence

In this section, we offer two examples of using the theory developed in the previous section. Figure 16 shows a histogram showing the predicted and simulated probability density for time to replenishment.

The spike shown at the beginning of Figure 16 for the prediction curve represents Pr(Tr=0), and the remainder of the curve is $f_{Tr}(t)$.

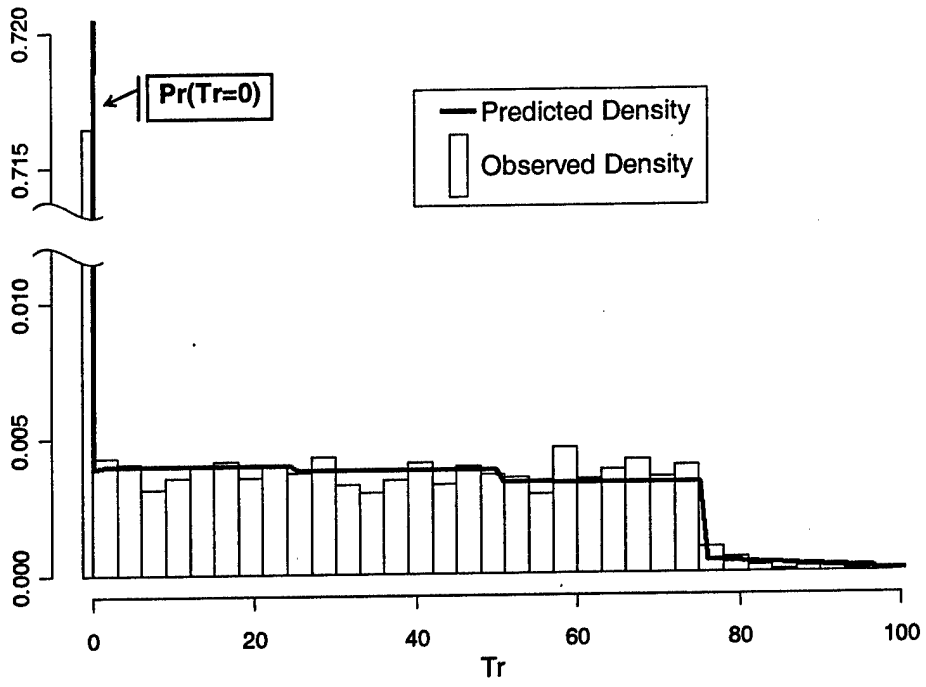


Figure 16: $f_{Tr}(t)$ Predicted Versus that Observed Through Simulation

The next three figures (using the same parameters as Figure 8 where T_p is set to 1) show predictions versus simulations for the average latency (E[W]), average queuing time (E[Q]), and average service time ($E[S_s]$) respectively. The predictions for the average queuing time appear to be fairly accurate. The predictions for average service time appear to be accurate until U_p exceeds 0.65. As discussed in Section 4.4.3, as the average duration of the busy period increases (which happens as U_p increases) and starts to approach T_{ss}, our approach for calculating Pr(Tr=0) becomes less accurate, which, in turn, affects the accuracy of $E[S_s]$.

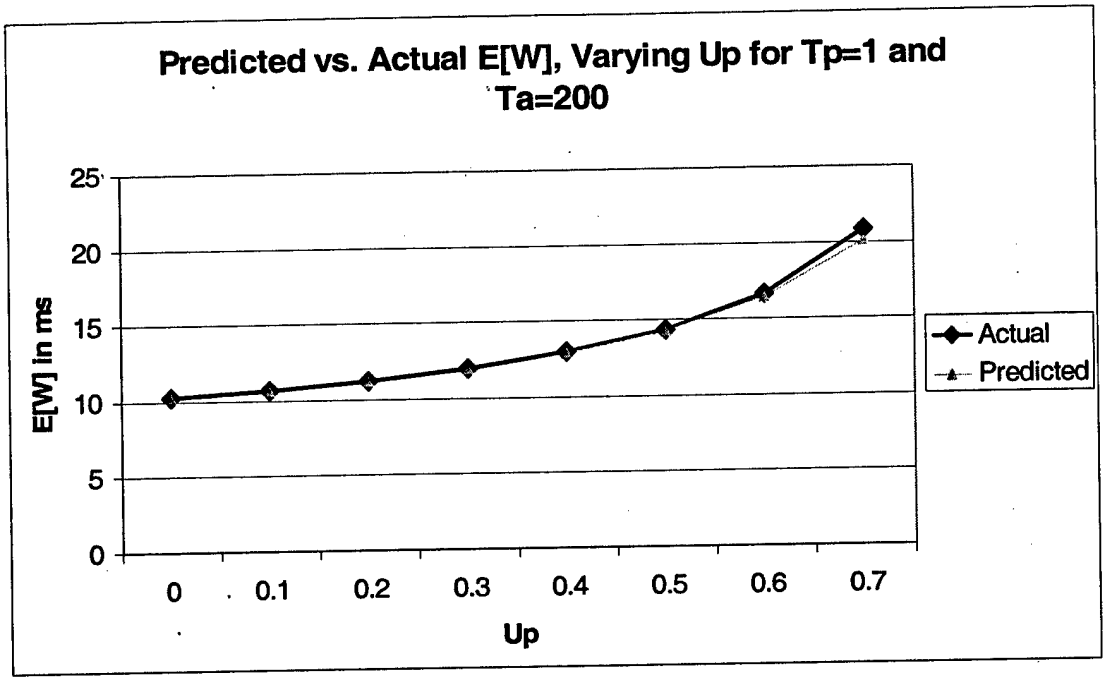


Figure 17: E[W] Predicted Versus that Observed Through Simulation

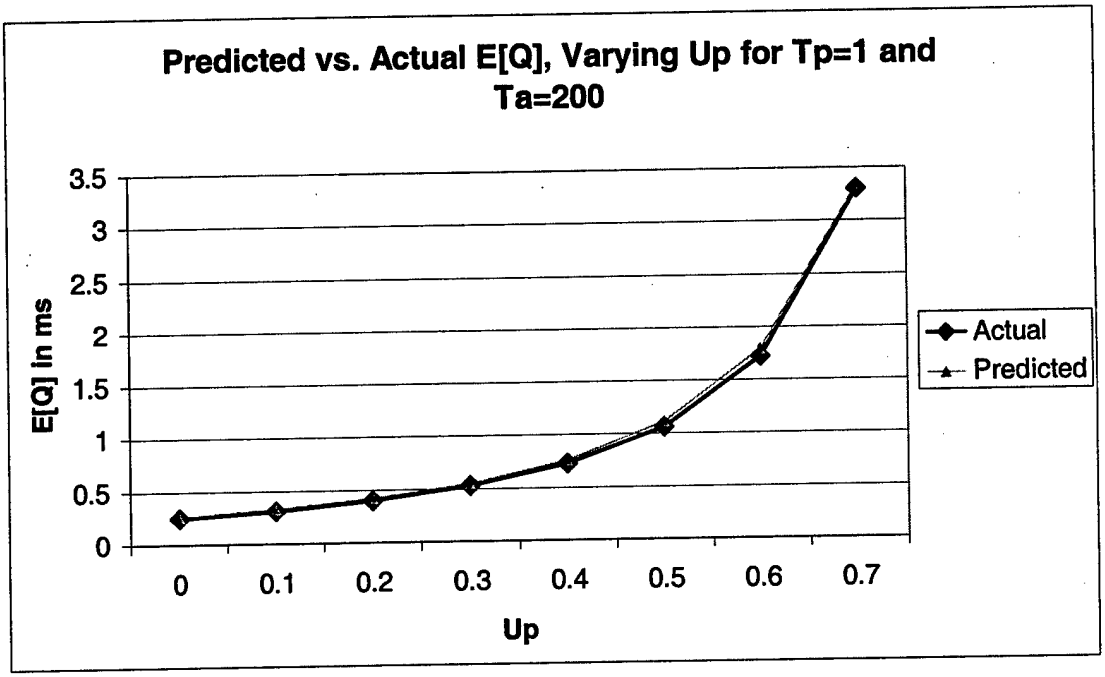


Figure 18: E[Q] Predicted Versus that Observed Through Simulation

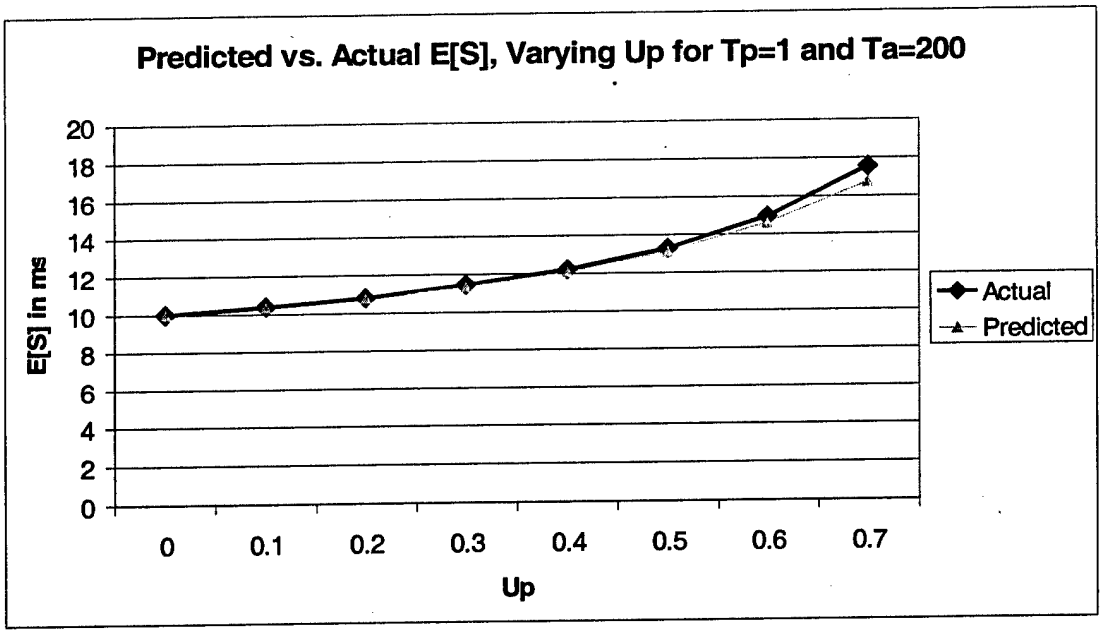


Figure 19: E[S_s] Predicted Versus that Observed Through Simulation

4.5 Single-Subtask Assemblies

In this section, we look at assemblies with three periodic tasks and an aperiodic task managed by a sporadic server. Anticipating applying our theory to more general assemblies, we make some observations about multitask assemblies.

Varying utilization equally. For this case, there is an aperiodic task and multiple periodic tasks between which the periodic utilization is evenly divided. The average latency for the multi-periodic case should fall within the extremes of the single periodic cases. The parameters for this graph are T_a=200, S_a=S_{ss}=10, and T_{ss}=100, while T_p and U_p vary accordingly. This case is shown in Figure 20 except for when U_p is large: we intend to investigate this further.

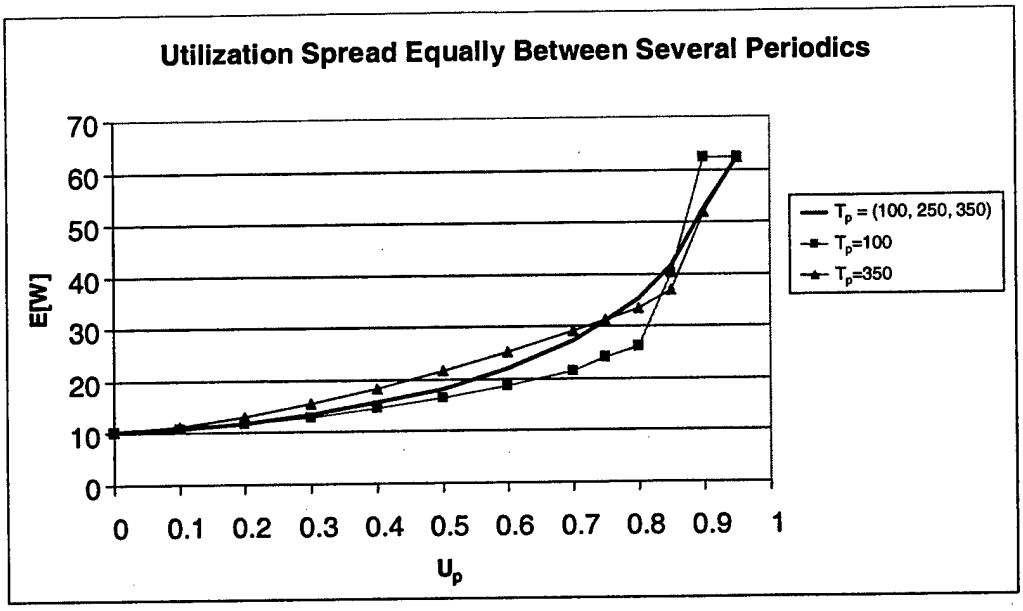


Figure 20: Multi-Periodic Example—Utilization Evenly Divided

Varying utilization unequally. In the previous case, there was a single aperiodic task and three periodic tasks. However, in this case, the total utilization of the periodics (which varies from 0 to .9) is spread unevenly among the three periodic tasks. In the legend of Figure 21, ".8 .1 .1" means that for a utilization level of U_p, .8* U_p is accorded to the task whose period is 100, .1* U_p is accorded to the task whose period is 250, and .1*U_p is accorded to the task whose period is 350. Additionally, we plot the single periodic case for T_p=100 and T_p=350. Again, except for relatively large values of U_p, the single periodic cases create an envelope around the multi-periodic cases.

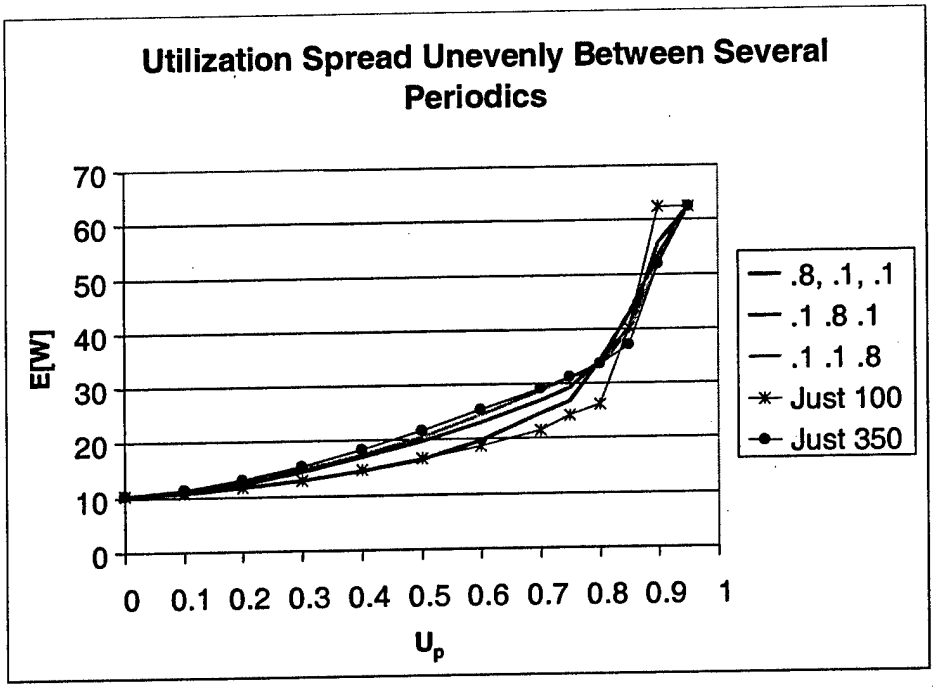


Figure 21: Multi-Periodic Example—Utilization Unevenly Divided

4.6 Multi-Subtask Assemblies

We took the multi-periodic case shown in Figure 20 and turned each periodic task into one that had multiple subtasks with arbitrary priorities. The graph in Figure 22 shows the results.

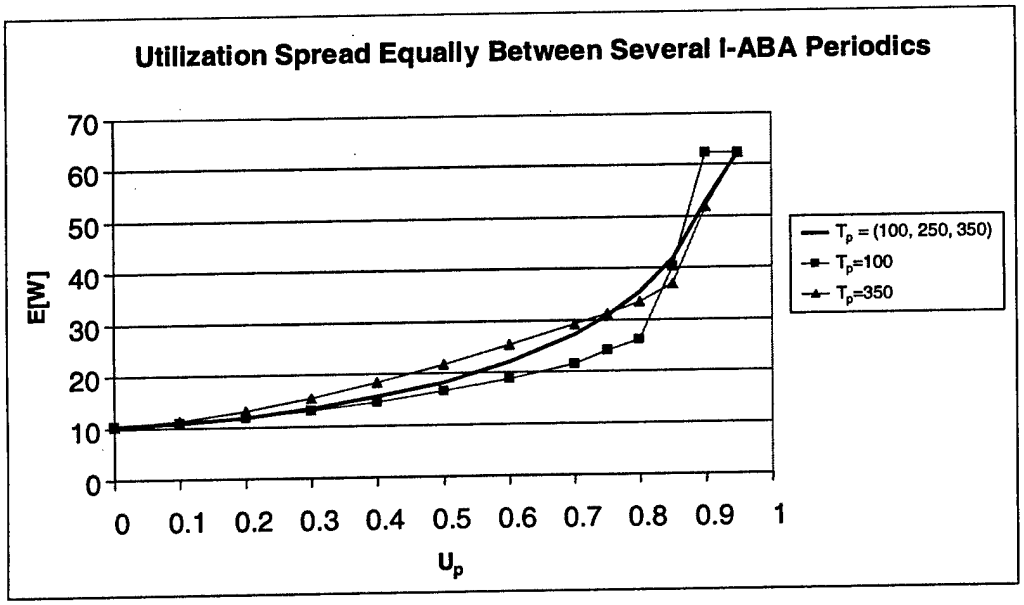


Figure 22: Multi-Periodic Example with Multiple Subtasks

Notice that the graphs in Figure 22 and Figure 20 are identical. As long as the system is work conserving (i.e., it continues to do available work without idling), the periodics' priority and subtask structures do not influence the average latency of the aperiodics. This lack of influence occurs because the periodic subassembly's priority and subtask structure do not influence when background is available.

This allows an arbitrarily complex periodic task to be simplified to an equivalent periodic single-subtask task.

4.7 Observations on the No-Background Case

Notice that in Figure 8, Figure 21, and Figure 22, the length of the periodic task's period influences when the aperiodic task's average latency reaches the point of so-called "no background." To understand this situation, consider the two extreme cases: infinitesimal periods (continuous background) and very large periods.

In the continuous background case, S_q approaches T_{ss} as U_p increases. When $U_p=1-S_{ss}/T_{ss}$, S_q (which is equal to $\hat{S}_a$) is equal to T_{ss}. This is the maximum possible value for S_q (as guaranteed by the sporadic server). Therefore, the continuous background case reaches a state of "no background" when $U_p=1-S_{ss}/T_{ss}$. Notice that while the aperiodic events cannot execute in background, the processor is not necessarily fully utilized. For example, if $S_{ss}=10$, $T_{ss}=100$, $S_a=10$, $T_a=200$, and $U_p=.9$, then $S_q=T_{ss}=100$ and $\rho=S_q/T_a=100/200=0.5$. This means that, on average, only half of the "degraded processor" is actually being utilized. In other words, only half of the unused periodic utilization is used by the aperiodic task.

As the length of the periodic tasks' period increases, the aperiodic task is able to use this unused capacity. When the period of the periodic tasks is very large, the aperiodic task can execute both within the sporadic server and in the large windows of periodic idleness.

5 Application of the Theory

In this section, we describe a collection of heuristics that encompass much of the observations and analysis performed in Section 4. We then describe a model problem from the domain of robotics and show how to apply these heuristics derived from the property theory to the robotics model problem.

5.1 Reasoning Heuristics

Much of this report has focused on providing a theoretical foundation for the continuous background case with a single periodic task. However, important conclusions can also be derived from empirical evidence. Sections 4.5 and 4.6 showed that with some targeted simulations we can get a good understanding of aperiodic latency for a very general periodic task set. Based on our theoretical and empirical understanding, we generated the following list of heuristics:

- **H1:** For a given aperiodic service time (S_a) and interarrival interval (T_a), the best-case average latency occurs when there are no periodics (U_p=0). The latency for this case is predictable by Equation (4).
- **H2:** For a given aperiodic service time and interarrival interval, the worst-case average latency occurs when the periodic utilization is large enough so that aperiodics execute only within the sporadic server (no-background case). The latency for this case is predictable by Equation (6), where $S_q = T_{ss}$.
- **H3:** For the continuous background case, given U_p, E[Q] can be predicted accurately by using Equation (9) and letting $S_q = \hat{S}_a$. E[S_s] can be approximated by realizing that it is a weighted average of S_a and $\hat{S}_a$ and therefore is between those two extremes. As U_p gets larger, $\hat{S}_a$ starts to approach T_{ss}, so there is very little room for background processing. In this case (even though E[Q] increases), E[S_s] approaches S_a. H3 applies to cases in which U_p is greater than 0 and less than 1-S_{ss}/T_{ss}.
- **H4:** For very large periodic periods, average latency as a function of U_p appears to be approximately the convex combination of the no-periodics (NP) and no-background (NB) cases. H4 applies to cases in which U_p is greater than 0 and less than 1-ρ. Therefore, in this case

$$E[W] = \left(\frac{E[W_{NB}] - E[W_{NP}]}{1-\rho} \right) U_p + E[W_{NP}], where$$

$$E[W_{NP}] = \left(\frac{\rho}{1-\rho} \right) \frac{E[S_a^2]}{2E[S_a]} + E[S_a], and$$

$$E[W_{NB}] = \left(\frac{\hat{\rho}}{1-\hat{\rho}} \right) \frac{E[\hat{S}_a^2]}{2E[\hat{S}_a]} + E[S_a], where\ \hat{\rho} = \frac{\hat{S}_a}{T_a}\ and\ \hat{S}_a < T_a\ and\ \hat{S}_s = T_{ss}$$

The observations gleaned from the curves generated for average-case latency for aperiodic events served by a sporadic server (see Figure 8) provided significant understanding of the timing-related behavior of such events. Figure 23 below represents an abstraction of Figure 8 with the heuristics overlaid to illustrate how these heuristics support predicting aperiodic latency.

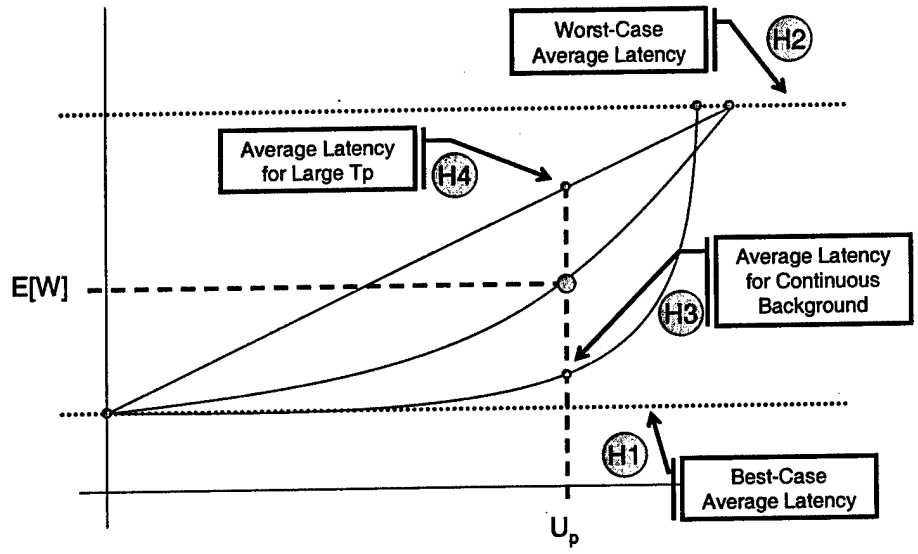


Figure 23: Heuristics Applied to the Curves

Notice in Figure 23, H1 (the best-case average latency) and H2 (the worst-case average latency) serve to bound the aperiodic average latency, E[W]. Also notice that for a specific value of U_p, H3 and H4 seem to provide bounds for E[W]. E[W], then, will fall between the best-case and worst-case average latency dictated by H1 and H2, within the region further defined by H3 and H4 for a specific U_p.

In the next section, we apply these heuristics to a model problem.

5.2 A Robotics-Based Model Problem

To demonstrate the analytical and empirical foundations established in this paper, we apply them to a model problem [Hissam 04] representative of a design problem posed for the ABB industrial robotics product line.

The model problem expresses the high-level task structure used to convey robot movement commands through a series of queues to ultimately control the various axes of a robot's arm(s). The model problem permits the incorporation of additional end-user tasks (or extensions) in the controller similar to the addition of a third-party device driver in an OS. It is this extensibility that motivates the use of this performance theory. That is, an extension will be either periodic or aperiodic by nature. The reasoning framework discussed by Hissam and colleagues [Hissam 02] can be applied to predict the average latency of periodic extensions (see Section 2.2). The analytical and empirical foundations introduced in this report can be applied to predict the timing behavior of aperiodic extensions.

In summary, the robotics-based model problem has

- periodic and aperiodic tasks
- tasks with hard deadlines and average-case latency requirements
- tasks (for example, controller extensions) whose behavior must be both predictable and predictably invasive on other periodic tasks with hard deadlines
- requirements for predicting deadline miss rates

In the remainder of this section, we apply the performance theory in this paper to predict the average-case latency of an aperiodic task within the robotics-based model problem.

5.2.1 Tasks in the Model Problem

Figure 24 provides a schematic of the open robotics model problem. In this discussion, we simplify the model further by concentrating on only one task set—A-B-C (controlling a robot with only one arm)—and one "plug-in"—task M.

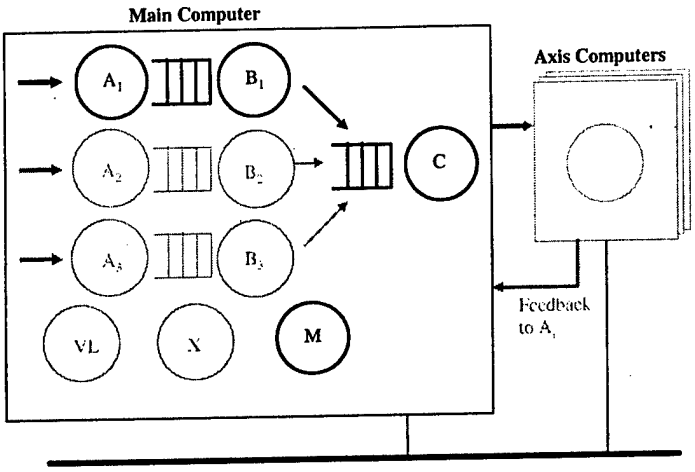


Figure 24: Tasks in the Robotics Model Problem

Tasks A, B, and C each convey commands through a series of queues. Task M represents a third-party task extension to the robotics controller.

Table 2 shows the applicable task performance specifications of the tasks in Figure 24.

Table 2: Performance Description of Model Problem Tasks

Task	Priority	Arrivals	Execution Time
A_1	Low	Exponentially distributed with mean 75 ms	Exponentially distributed with mean 9 ms
B_1	High	Constant 24ms	Uniform 1-2 ms
C	Very High	Constant 4ms	Uniform 0.5-1 ms
M	Med	Exponentially distributed with mean 100 ms	Uniform 15-25 ms

5.2.2 Analysis Setup

We are interested in two questions in particular:

1. Will including Task M cause Tasks A, B, and C to miss their deadlines?
2. What is the average-case latency of events handled by Task M?

To answer these questions, we take a closer look at Tasks A, B, and C.

Task A is a low-priority task that is handling a stream of aperiodic arrivals. Each aperiodic arrival is broken down into a sequence of subitems and placed on the queue between Task A and B.

Task B continues to periodically process a subitem from the Task A-B queue, further decomposes that subitem, and places the resulting decompositions on the queue between Tasks B and C.

The period of Task B (i.e., 24 ms) is 6 times the period of Task C (i.e., 4 ms). That is true because for every subitem processed by Task B (generating 6 microcoordinates), Task C will consume a microcoordinate from the Task B-C queue and send it to the Axis computer (which itself takes about 1 ms). Task C then can consume all six microcoordinates on the Task B-C queue within the period of Task B.

Ideally, Task A should never allow the queue between Task A and B (i.e., the Task A-B queue) to empty. Unfortunately, Task A is only given low priority in the controller because, under certain conditions, Task A may take an inordinate amount of CPU time to perform its item decomposition. If Task A were assigned a higher priority relative to Task B or Task C, Task A could cause either of those tasks to miss its deadline. At low priority, Task A is assured not to interfere with the deadlines of Tasks B and C. However, with the inclusion of an extension to the controller, Task A could be starved to the point that it could inadvertently starve the Task A-B queue.

To solve this conundrum and ensure that Task A does have the opportunity to put at least one subitem on the Task A-B queue within the period of Task B, Task A can be converted to use the SSSA.[20] In this case, Task A can be given just enough execution time to prevent the Task A-B queue from being emptied. Further, given that Task A is following the SSSA, Task A can now be treated like a periodic task.

Understanding the interactions between Tasks A, B, and C and treating Task A as a periodic task allow us to apply the results of Section 4.5 that deal with single-subtask assemblies. This analysis asserts that assemblies with multiple periodic tasks can be analyzed by considering single-task assemblies whose utilization (U_p) is the same as that of more complex assemblies and by varying the periods between the smallest and largest periodic periods.

[20] This point is another design issue that Hissam and Klein address in another report [Hissam 04].

This means that the task set (A-B-C) in Figure 24 can be combined into a single periodic task for which T_p = 24 ms (Task B's period) and the execution time is approximately 10 ms.[21] Figure 25 shows the final periodic single subtask created to support the analysis of the robotics model problem.

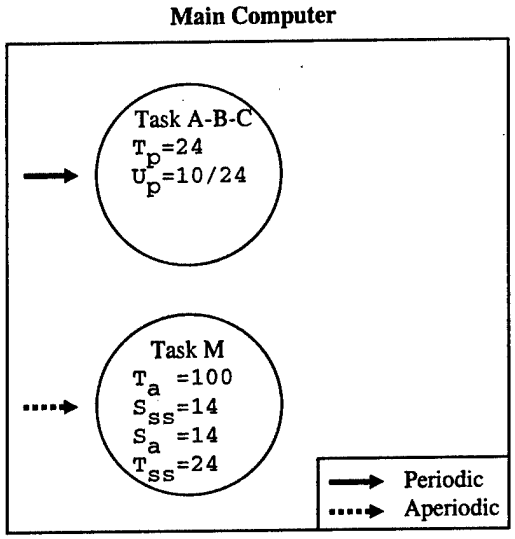


Figure 25: Analytic Representation of the Robotics Model Problem

The performance parameters for the extension (i.e., Task M) can be taken mostly from Table 2. Task M is an aperiodic task managed by the SSSA. Its interarrival interval (T_a) is 100 ms. The replenishment period for Task M is not specified; however, T_{ss} must be $\geq T_p$, otherwise the aperiodic task might be able to preempt the periodic task more than once during its period and put the periodic's deadline at risk. T_{ss}, then, can (at best) take on the value of 24 ms.

The last two performance parameters for Task M are S_{ss} (budget) and S_a (execution time). The upper limit for S_{ss} (given that T_{ss}=24 ms) is determined by the total utilization of the two tasks in Figure 25. The highest value for S_{ss} must be 14 ms, resulting in a total utilization for Task M being 14/24 (S_{ss}/T_{ss}). Finally for S_a, Section 4 states in the governing assumptions that S_a=S_{ss}; we will assume that Task M's execution time is a constant 14 ms and perform our analysis from this point.

5.3 Preserving Periodic Deadlines

The first question is whether Task M will cause the A-B-C task set to miss its deadline. This set executes for 10 ms every 24 ms with a deadline at the end of its period. In the worst case,

[21] Assuming the worst case, execution times for Tasks B and C are 2 ms and 6 ms, respectively (from Table 2—recall that Task C will execute for 1 ms, 6 times during Task B's 24-ms period); Task A's execution time is based on the time it takes to produce 1 subitem from its input stream: approximately 1-2 ms. The sum of these approximate times is just under 10 ms.

Task M will preempt the A-B-C task set once for 14 ms, implying a worst-case latency of 24 ms for the task set and, hence, guaranteeing the set's deadline.

5.4 Predicting Average-Case Latency

The second question is to predict the average-case latency of the aperiodic extension (Task M). In this section, we apply heuristics from Section 5.1.

Heuristic H1—best-case average latency: $S_a = 14$; $T_a = 100$; $\rho = S_a / T_a = 14/100 = 0.14$. Solving for Equation (4), we get the following:

$$\begin{aligned} E[W] &= \left(\frac{\rho}{1-\rho}\right)\left(\frac{E[S_a^{\,2}]}{2E[S_a]}\right) + E[S_a] \\ &= \left(\frac{.14}{.86}\right)\left(\frac{14^2}{2 \bullet 14}\right) + 14 \\ &= 15.13953 \end{aligned} \tag{39}$$

Heuristic H2—worst-case average latency: $E[S_q] = T_{ss}$; $E[S_s] = S_a$; $\rho_q = S_q / T_a = 24/100 = 0.24$. Solving for Equation (6), we get the following:

$$\begin{aligned} E[W] &= \left(\frac{\rho_q}{1-\rho_q}\right)\left(\frac{E[S_q^2]}{2E[S_q]}\right) + E[S_s] \\ &= \left(\frac{.24}{.76}\right)\left(\frac{24^2}{2 \bullet 24}\right) + 14 \\ &= 17.78947 \end{aligned} \tag{40}$$

Heuristic H3—bound on average-case latency for continuous background for a specific U_p: Since $U_p = 1 - (S_{ss} / T_{ss})$ in this model problem, H3 cannot be used.

Heuristic H4—bound on average-case latency for large T_p for a specific U_p: $\rho = S_a / T_a = 14/100 = 0.14$. Using the equation from H4, we get the following:

$$E[W] = \left(\frac{E[W_{NB}] - E[W_{NP}]}{1-\rho} \right) U_p + E[W_{NP}], where$$

$$E[W_{NP}] = \left(\frac{\rho}{1-\rho} \right) \frac{E[S_a^2]}{2E[S_a]} + E[S_a], and$$

$$E[W_{NB}] = \left(\frac{\hat{\rho}}{1-\hat{\rho}} \right) \frac{E[\hat{S}_a^2]}{2E[\hat{S}_a]} + E[S_a], where\ \hat{\rho} = \frac{\hat{S}_a}{T_a}\ and\ \hat{S}_a < T_a\ and\ \hat{S}_s = T_{ss}$$

For $E[W_{NP}]$, we can use the result calculated for H1 above: $E[W_{NP}] = 15.13953$. For $E[W_{NB}]$, we can use the result calculated from H2 above: $E[W_{NB}] = 17.78947$. Then, solving for E[W], we get the following:

$$\begin{aligned} E[W] &= \left(\frac{E[W_{NB}] - E[W_{NP}]}{1-\rho} \right) U_p + E[W_{NP}] \\ &= \left(\frac{17.78947 - 15.13953}{.86} \right) (10/24) + 15.13953 \\ &= 16.42342 \end{aligned}$$

Heuristic H1 tells us that for the model problem, we can expect the average latency to be no better than 15.13953 ms. Further, H2 tells us that we can expect the average latency to be no worse than 17.78947 ms. We can refine the lower bound for $U_p = 10/24 = 0.4166$ (see Figure 26) by applying H4; the result is 16.42342 ms.

By overlaying the results from the heuristics just computed to a plot of curves representing latencies observed through simulation, it is possible to check the heuristics. Figure 26 was created through simulation. All the performance parameters from Figure 25 were used in the simulation except for T_p and U_p, which varied. The simulation was run for 13 values of U_p (specifically 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, and 0.95) for each T_p shown in the legend of the figure. The average latency for the aperiodic Task M was computed, recorded, and plotted based on the specific U_p and T_p for that simulation.

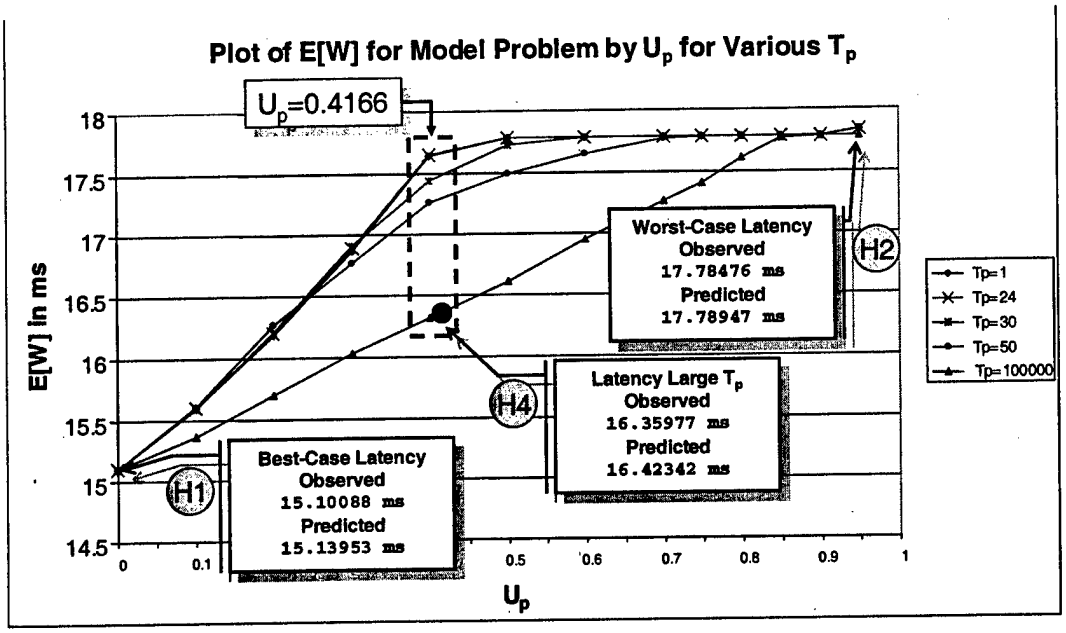


Figure 26: Latency Observed for the Model Problem for Various T_p Values

As described in Section 4.7, the no-background state is reached at different periodic utilizations for different periodic periods. For this model problem, the no-background state is reached at a much smaller utilization for very small periods than for very large ones. The net effect is that the above graph almost looks like an inverted version of Figure 8.

Table 3 shows the side-by-side comparisons of the heuristics calculated and the averages observed through simulation in Figure 26.

Table 3: Comparison of Prediction Heuristics and Simulation Curves

	Computed Heuristic (in ms)	**Observed in Simulation (in ms)**
H1: Best-Case Average Latency	15.13953	15.10088
H2: Worst-Case Average Latency	17.78947	17.78476
H4: Bound for Large T_p	16.42342	16.35977

Because the model problem represents an extreme case where $U_p = 1 - S_{ss} / T_{ss}$ (i.e., the no-background case), the heuristic H2 should offer an accurate prediction. The value computed for H2 was compared to the average latency observed in many simulations of the model problem. These results are reported in Table 4. The prediction of 17.78947 ms was found to be within 2 standard errors of the observed latency of 17.79473 ms.

Table 4: Predicted and Actual Average-Case Latency for Task M

Basic Statistics	**Value**
Samples (n)	1035
Average Aperiodic Latency Observed	17.79473 ms
Standard Deviation (σ)	0.12574 ms
Predicted Aperiodic Latency (E[W])	17.78947 ms
Error	0.00526
Standard Error	0.00391
2 x Standard Errors	0.00782

6 Conclusions

This report documents the development and application of a theory for predicting the average latency of aperiodic tasks that execute under the management of a sporadic server. The notion of a sporadic server was invented many years ago, but this is the first time a detailed queueing-theoretic analysis has been performed. Such analysis represents a step in the direction of creating a comprehensive set of performance reasoning frameworks that includes using RMA for a deterministic deadline analysis, queueing theory for average latency analysis, and a new theory known as real-time queueing theory (RTQT) [Doytchinov 01] for probabilistic analysis of deadline miss rates.

This report focused on task sets with the following characteristics:

- The assemblies are confined to a single processor.
- Each periodic event, whether clock or message based, is handled by one task (or a sequence of tasks). Each periodic task has an associated period and an execution time (or sequence of execution times).
- All aperiodic events are funneled through a sporadic server.
- The sporadic server runs at the highest priority in the system and is characterized by an execution budget and a replenishment period.
- The service time for each aperiodic event is constant and equal to the execution budget of the sporadic server.
- The aperiodic arrivals arrive according to an exponential distribution with a specified mean interarrival interval. We only consider a single stream of aperiodic events.
- Aperiodic events are allowed to use the CPU when either the sporadic server has sufficient budget or the periodics are idle.

The analytical and empirical results described in this paper are applicable to a very large spectrum of assemblies. In fact, the periodic assemblies can be arbitrarily complex.

6.1 Future Work

While we have gained many insights from examining a specific class of assemblies, our objectives are to solidify the theoretical foundations for what we have discussed in this paper and to relax the assumptions, investigating a more general set of assemblies.

- We would like to gain a better understanding of the "middle period cases." We understand much about the cases where the period of the periodics is very small (relative to S_{ss}) and very large. However, we have only just begun to understand the midrange periods.
- We need to experiment with multi-periodic assemblies with a larger number of tasks. We suspect that we can create situations in which the average latency of the aperiodics is even better than the lower bound we observed in multi-periodic cases discussed in this report.
- We have enumerated several places where our mathematical modeling needs to be improved including accounting for length biasing, refining our blackout distributions, and accounting for when many replenishments occur during a busy period.
- We intend to combine RTQT with the sporadic server for predicting the probability of missing deadlines.
- Currently, arrivals and service time distributions are constrained to exponential and constant distributions, respectively. Heavy traffic approximations might allow us to acquire an analytic understanding while relaxing these restrictions. In any case, we will perform empirical studies.
- Currently the sporadic server is confined to $S_{ss} = S_a$, S_a is confined to being constant, and the sporadic server must execute at the highest priority. We will investigate relaxing these restrictions and whether, with a more general sporadic server capability, we may have more control over the average latency of the aperiodics.
- So far, we have only investigated the uniprocessor case. In the future, we plan to investigate the distributed problem.
- Earlier, we briefly investigated applying RTQT in a fixed-priority setting. We will continue that investigation.
- We plan to implement an earliest deadline first (EDF) capability in our runtime infrastructure—a natural setting for applying RTQT.

We plan to pursue the above areas with an eye towards their practical application.

Bibliography

URLs are valid as of the publication date of this document.

[Cinlar 97] Cinlar, E. *Introduction to Stochastic Processes*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1997.

[Doytchinov 01] Doytchinov, B.; Lehoczky, J. P.; & Shreve S. "Real-Time Queues in Heavy Traffic with Earliest-Deadline-First Queue Discipline." *Annals of Applied Probability 11*, 2 (May 2001): 332-378.

[Gonzalez Harbour 91] Gonzalez Harbour, M. & Sha, L. *An Application-Level Implementation of the Sporadic Server* (CMU/SEI-91-TR-026, ADA242129). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991. http://www.sei.cmu.edu/publications/documents/91.reports/91.tr.026.html

[Hissam 02] Hissam, S.; Hudak, J.; Ivers, J.; Klein, M.; Larsson, M.; Moreno, G.; Northrop, L.; Plakosh, D.; Stafford, J.; Wallnau, K.; & Wood, W. *Predictable Assembly of Substation Automation Systems: An Experiment Report, Second Edition* (CMU/SEI-2002-TR-031, ADA418441). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tr031.html

[Hissam 04] Hissam, S. & Klein, M. *A Model Problem for an Open Robotics Controller* (CMU/SEI-2004-TN-030). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tn030.html

[Klein 93] Klein, M.; Ralya, T.; Pollak, B.; Obenza, R.; & Gonzalez Harbour, M. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993. http://www.sei.cmu.edu/publications/books/other-books/rma.hndbk.html

[Kleinrock 75] Kleinrock, L. *Queueing Systems Volume 1: Theory*. New York, NY: Wiley, 1975 (ISBN: 0-471491-10-1).

[R Development Core Team 04] R Development Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2004 (ISBN 3-900051-00-3). http://www.R-project.org

[Ross 96] Ross, Sheldon. *Stochastic Processes Second Edition*. New York, NY: John Wiley & Sons, Inc., 1996 (ISBN: 0-471-12062-6).

[Shi 01] Shi, W. *Implementation and Performance of POSIX Sporadic Server Scheduling in RTLinux* (TR-010602). Tallahassee, FL: Florida State University, 2001. http://websrv.cs.fsu.edu/research/reports/TR-010602.ps

[Sprunt 89] Sprunt, B.; Sha, L.; & Lehoczky, J. *Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System* (CMU/SEI-89-TR-11, ADA211344). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1989. http://www.sei.cmu.edu/publications/documents/89.reports/89.tr.011.html

[Wallnau 03] Wallnau, K. *Volume III: A Technology for Predictable Assembly from Certifiable Components* (CMU/SEI-2003-TR-009, ADA413574). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr009.html

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2004	Final

4. TITLE AND SUBTITLE
Performance Property Theories for Predictable Assembly from Certifiable Components (PACC)

5. FUNDING NUMBERS
F19628-00-C-0003

6. AUTHOR(S)
Scott Hissam, Mark Klein, John Lehoczky, Paulo Merson, Gabriel Moreno, Kurt Wallnau

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2004-TR-017

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-2004-017

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE
70

13. ABSTRACT (MAXIMUM 200 WORDS)

This report develops a queueing-theoretic solution to predict, for a real-time system, the average-case latency of aperiodic tasks managed by a sporadic server. The report applies this theory to a model problem drawn in the domain of industrial robot control. In this model problem, a controller with hard periodic deadlines is "open" to third-party plug-in extensions. The sporadic server is used to limit the invasiveness of aperiodic tasks on the controller's hard deadlines. The theory developed in this report is used to predict the average-case latency of a plug-in managed by a sporadic server.

14. SUBJECT TERMS
rate monotonic analysis, reasoning framework, real-time analysis, predictable assembly, sporadic server, real-time queueing, latency

15. NUMBER OF PAGES
70

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL